Fernando Orejas
Paul G. Spirakis
Jan van Leeuwen (Eds.)

LNCS 2076

Automata, Languages and Programming

28th International Colloquium, ICALP 2001
Crete, Greece, July 2001
Proceedings

Springer

Lecture Notes in Computer Science 2076
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Fernando Orejas Paul G. Spirakis
Jan van Leeuwen (Eds.)

Automata, Languages and Programming

28th International Colloquium, ICALP 2001
Crete, Greece, July 8-12, 2001
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Fernando Orejas
Univ. Politècnica de Catalunya, Departament de Llenguatges i Sistemes Informàtics
C/Jordi Girona Salgado 1-3, 08034 Barcelona, Spain
E-mail: orejas@lsi.upc.es

Paul G. Spirakis
University of Patras, Computer Technology Institute (CTI)
61 Riga Feraiou street, 26221 Patras, Greece
E-mail: spirakis@cti.gr

Jan van Leeuwen
Utrecht University, Institute of Information and Computing Sciences
Padualaan 14, 3584 CH Utrecht, The Netherlands
E-mail: jan@cs.uu.nl

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Automata, languages and programming : 28th international colloquium ;
proceedings / ICALP 2001, Crete, Greece, July 8 - 12, 2001. Fernando Orejas
... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ;
Milan ; Paris ; Singapore ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2076)
ISBN 3-540-42287-0

CR Subject Classification (1998): F, D, C.2-3, G.1-2, I.3

ISSN 0302-9743
ISBN 3-540-42287-0 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna
Printed on acid-free paper SPIN: 10781802 06/3142 5 4 3 2 1 0

Preface

The *28th International Colloquium on Automata, Languages and Programming* (ICALP 2001) was held July 8-12, 2001 in the Aldemar-Knossos Royal Village near Hersonissos on Crete, Greece. This volume contains all contributed papers presented at ICALP 2001, together with the invited lectures by Ahmed Bouajjani (Paris), Martin Große-Rhode (Berlin), Mogens Nielsen (Aarhus), and Ingo Wegener (Dortmund) and two of the keynote lectures, by Christos Papadimitriou and Boris Trakhtenbrot.

For almost 30 years now, ICALP has been the main annual event of the European Association for Theoretical Computer Science (EATCS). The ICALP program currently consists of track A: *Algorithms, Automata, Complexity, and Games* and track B: *Logic, Semantics, and Theory of Programming*.

In response to the Call for Papers, the program committee received 208 submissions: 162 for track A, 46 for track B. The committee met on March 23/24, 2001 in Barcelona and selected 80 papers for inclusion into the scientific program. The selection was based on originality, quality, and relevance to theoretical computer science. We wish to thank all authors who submitted extended abstracts for consideration, and all 366 referees who helped in the extensive evaluation process. The program committee for ICALP 2001 consisted of:

TRACK A

Maxime Crochemore (Marne-la-Vallée)
Leslie A. Goldberg (Warwick)
Mordecai J. Golin (Hong Kong)
Juraj Hromkovic (Aachen)
Guiseppe F. Italiano (Rome)
Viggo Kann (Stockholm)
Ludek Kucera (Prague)
Bill McColl (Oxford and Sychron Inc.)
David Peleg (Rehovot)
Jose Rolim (Geneva)
Peter Sanders (Saarbrücken)
Erik M. Schmidt (Aarhus)
Maria Serna (Barcelona)
Jack Snoeyink (Chapell Hill)
Athanasios K. Tsakalidis (Patras)
Jan van Leeuwen (Utrecht, *chair*)
Dorothea Wagner (Konstanz)

TRACK B

Samson Abramsky (Oxford)
Kim B. Bruce (Williamstown)
Stavros Cosmadakis (Patras)
Hartmut Ehrig (Berlin)
Javier Esparza (München)
Thomas A. Henzinger (Berkeley)
Jean-Pierre Jouannaud (Orsay)
Jose Meseguer (SRI Menlo Park)
Eugenio Moggi (Genova)
Ugo Montanari (Pisa)
Damian Niwinski (Warsaw)
Fernando Orejas (Barcelona, *chair*)
Catuscia Palamidessi (Penn State)
Andreas Podelski (Saarbrücken)
Hanne Riis Nielson (Lingby)

The *EATCS Best Paper Award* was given to William Hesse (Amherst MA, also *Best Student Paper*) and to Parosh Aziz Abdulla, Luc Boasson, and Ahmed Bouajjani (Uppsala/Paris) for their respective papers in ICALP 2001.

ICALP 2001 was a very special ICALP. Two other leading Computer Science conferences co-located with ICALP this time: the 13th Annual ACM Symposium

on Parallel Algorithms and Architectures (SPAA 2001), and the 33rd Annual ACM Symposium on Theory of Computing (STOC 2001), giving the joint participants an unprecedented opportunity in Europe to see the advances in a broad spectrum of foundational computer science research. It was the first time ever that STOC had been held outside of the USA.

During STOC 2001 and ICALP 2001 the following special events took place: (a) the *Turing Award Lecture* by Andrew C-C. Yao (Princeton), (b) the *EATCS Distinguished Service Award* for Corrado Böhm (Rome), (c) the *Greek Computer Society/CTI Prize* for Christos Papadimitriou (Berkeley) recognizing him as the most influential scientist of Greek origin for his contributions to the foundations of computer science, and (d) the *Gödel Prize 2001* awarded to S. Arora, U. Feige, S. Goldwasser, C. Lund, L. Lovasz, R. Motwani, S. Safra, M. Sudan, and M. Szegedy for their fundamental papers on PCP's and the complexity of approximation problems. Special attention was given to the 80th birthday of B.A. Trakhtenbrot (Tel Aviv).

Several high-level workshops were held as 'satellite events' of ICALP 2001, with Christos Zaroliagis (Patras) as coordinator. This included the following workshops: *Algorithmic Methods and Models for Optimization of Railways* (ATMOS 2001), *Böhm's Theorem: Applications to Computer Science Theory* (BOTH 2001), *Graph Transformation and Visual Modeling Techniques* (2nd Int. Workshop, GT-VMT 2001), and *Verification of Parameterized Systems* (VEPAS 2001). The scientific program of ICALP 2001 showed that theoretical computer science is a vibrant field, deepening our insights into the foundations and futures of computing and system design in many modern application areas.

The sponsors of ICALP 2001 included the Information Society DG of the European Commission, the Ministry of the Aegean, the Ministry of Culture, and the Ministry of Education and Religious Affairs of Greece, the Hellenic Pedagogical Institute, the Intracom Institute of Technology (IIT), CTI (Patras), and the following companies: Intracom SA, Intralot SA, Intrasoft SA, ALTEC SA, MLS Multimedia SA, OPAP SA, Pliroforiki Technognosia Ltd, 01 Pliroforiki SA, Rainbow Computer SA, and Systema Informatics SA.

We are very grateful to the Computer Technology Institute (CTI) of Patras University for supporting the organization of ICALP 2001. The organizing committee consisted of: C. Bouras (CTI), S. Bozapalidis (Thessaloniki), R. Efstathiadou (CTI), C. Kaklamanis (CTI), M. Mavronicolas (Cyprus), C. Nikolaou (ICS-FORTH Crete), S. Nikoletseas (CTI), P. Spirakis (CTI), A. Tsakalidis (CTI), S. Zachos (Athens), and C. Zaroliagis (CTI). We thank Henk P. Penning and Henk van Lingen, system administrators at the Institute of Information and Computing Sciences of Utrecht University, for their outstanding effort in designing and developing the CSCS electronic conference server for ICALP 2001.

July 2001

Fernando Orejas
Paul G. Spirakis
Jan van Leeuwen

Table of Contents

Keynote Papers

Invited Papers

Algebraic and Circuit Complexity

Algorithm Analysis

Approximation and Optimization

Complexity

Concurrency

Efficient Datastructures

Graph Algorithms

Language Theory, Codes and Automata

Model Checking and Protocol Analysis

Networks and Routing

Reasoning and Verification

Scheduling

Secure Computation

Specification and Deduction

Structural Complexity

Algorithms, Games, and the Internet (Extended Abstract)

Christos H. Papadimitriou

Computer Science Division, UC Berkeley, Berkeley, CA 94720, USA,
christos@cs.berkeley.edu,
http://www.cs.berkeley.edu/~christos

Over the past fifty years, researchers in Theoretical Computer Science have sought and achieved a productive foundational understanding of the von Neumann computer and its software, employing the mathematical tools of Logic and Combinatorics. The next half century appears now much more confusing (half-centuries tend to look like that in the beginning). What computational artifact will be the object of the next great modeling adventure of our field? And what mathematical tools will be handy in this endeavor?

The Internet has arguably surpassed the von Neumann computer as the most complex computational artifact (if you can call it that) of our time. Of all the formidable characteristics of the Internet (its size and growth, its almost spontaneous emergence, its open architecture, its unprecedented availability and universality as an information repository, etc.), I believe that the most novel and defining one is *its socio-economic complexity:* The Internet is unique among all computer systems in that it is built, operated, and used by a multitude of diverse economic interests, in varying relationships of collaboration and competition with each other. This suggests that the mathematical tools and insights most appropriate for understanding the Internet may come from a fusion of algorithmic ideas with concepts and techniques from Mathematical Economics and Game Theory (see [3,5] for two excellent introductions in the respective subjects, and see the web site www.cs.berkeley.edu/~christos/cs294.html for many additional references of work in this interface.)

This talk is a survey of some of the many important points of contact between Game Theory and Economic Theory, Theoretical CS, and the Internet.[1]

Nash Equilibrium. Game theory was founded by von Neumann and Morgenstern (in fact, about the same time von Neumann designed the EDVAC...) as a general theory of rational behavior. The Nash equilibrium (definition omitted here) is the predominant concept of rationality in Game Theory; it is also a most fundamental computational problem whose complexity is wide open: Is there a polynomial algorithm which, given a two-person game with a finite strategy space, computes a mixed Nash equilibrium? Together with factoring, *the complexity of finding a Nash equilibrium is in my opinion the most important concrete open question on the boundary of P today.*

[1] Invited talk presented at a joint session of ICALP 2001 and STOC 2001. The full version of this paper appeared in the Proceedings of STOC 2001. Research supported by the National Science Foundation of the U.S.A.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 1–3, 2001.

In relation to the Internet, it would be most interesting to develop a game-theoretic model of Internet congestion control from which (an appropriate approximation of) TCP/IP emerges as a Nash equilibrium.

Coalitional Games. A coalitional game with n players is an increasing function $v : 2^{[n]} \mapsto \Re_+$; that is, a specification of the amount that each coalition of players "deserves." The fundamental issue in coalitional games is deciding whether a proposed allocation to the players, a vector $x \in \Re_+^n$ with $x[n] = v([n])$, is a "fair" way for the n players to split the loot in $v([n])$. A chief notion of fairness (among many others) is the *core:* A vector $x \in \Re_+^n$ with $x[[n]] = v([n])$ is in the core if $x[S] \geq v([S])$ for all S.

We can model the high-level operation of the Internet (the interaction of the "autonomous systems" that run it) as a coalitional game, as follows: We are given a graph with n nodes (the autonomous systems); an $n \times n$ symmetric traffic matrix F, where f_{ij} is the total traffic requirements between customers of i and customers of j; and a capacity c_i for each node (a simplification attempting to capture the capacity of i's subnetwork to carry traffic). If S is a set of nodes, consider the subgraph induced by S as a multicommodity network with node capacities and commodity requirements given by the entries of F; let $v(S)$ be the maximum total flow in this network —notice that this defines a coalitional game.

The key problem here is this: Is there an optimum solution in the multicommodity flow problem for the overall network, achieving a flow matrix $F' \leq F$, such that the corresponding payoffs for the nodes $x_i = \sum_j f'_{ij}$ are in the core of the coalitional game v (or abide by one of the other notions of fairness mentioned above).

The Price of Anarchy. There is no central authority that designs, engineers and runs the Internet.[2] But what if there were such master puppeteer, a benevolent Internet dictator who, for example, micromanaged its operation, allocating bandwidth to flows so as to maximize total user satisfaction? How much better would the Internet run? *What is the price of anarchy?*

This question was posed (and partially answered in the restricted context of a network consisting of two nodes and parallel edges) in [2]. More recently, [6] showed that, in the context of a general multicommodity flow network in which message delays increase with edge congestion while flows choose paths so as to minimize delay, the price of anarchy is *two* (more precisely, the anarchistic solution is no worse than the optimum solution with double the bandwidth).

But, of course, in today's Internet, flows cannot choose shortest paths. In the Internet, routers direct traffic based on local information, users respond to delay patterns by modifying their traffic, and network providers throw bandwidth at the resulting hot spots. How does this compare in efficiency with an ideal, *ab initio* optimum network design? *What is the price of the Internet architecture?*

[2] Recall David Clark's famous maxim: "We reject kings, presidents and voting. We believe in rough consensus and running code."

Mechanism Design. If Game Theory strives to understand rational behavior in competitive situations, the goal of Mechanism Design (an important and elegant research tradition, very extensive in both scope and accomplishment, and one that could alternatively be called "inverse game theory") is even grander: Given desired goals (such as to maximize a society's total welfare), design a game (strategies and payoffs) in such a clever way that individual players, motivated solely by self-interest, end up achieving the designer's goals. There have been recently interesting interactions between this fascinating area and Theoretical CS, see e.g. [4,1], and further opportunities abound.

Rough Markets. The famous Arrow-Debreu Theorem states that, under reasonable conditions, in any market there is always a set of prices that clears the market (agents optimizing their basket end up buying a total amount of each good which exactly equals the sum of the endowments that each agent brought in the market). But if the goods are integer-valued, then such an equlibrium may not exist. In recent joint work with Deng Xiaotie, we prove that it is NP-hard to tell if a price equilibrium exists even for very simple discrete markets; however, a price vector that clears markets *approximately on the average* (definition omitted) does exist and can be found in polynomial time —if the number of goods is bounded.

Finally, three more areas of contact between Theoretical Computer Science and Economics are discussed in the full paper: Economic aspects of **privacy** (algorithmic problems involved in computing the fair royalty for private information in various contexts), of **clustering** (how economic considerations can be a guide in the chaos of clustering criteria), and of **the web graph** (can the world-wide web's economic aspects explain its peculiar structure as a graph?).

References

1. J. Feigenbaum, C.H. Papadimitriou, S. Shenker, "Sharing the cost of multicast transmissions," *Proc. STOC 2000.*
2. E. Koutsoupias, C.H. Papadimitriou, "Worst-case equilibria," *Proc. STACS 1998.*
3. A. Mas-Colell, M.D. Winston, J.R. Green *Microeconomic Theory*, Oxford University Press 1995.
4. N. Nisan, "Algorithms for selfish agents – Mechanism design for distributed computation," *Proc. STACS 1999.*
5. M.J. Osborne, A. Rubinstein, *A Course in Game Theory*, MIT Press, 1994.
6. T. Roughgarden, E. Tardos, "How Bad is Selfish Routing? (Extended Abstract)," *Proc. FOCS 2000.*

Automata, Circuits, and Hybrids: Facets of Continuous Time

Boris A. Trakhtenbrot

School of Computer Science
Tel Aviv University
Tel Aviv 69978, Israel
trakhte@post.tau.ac.il

1 Introduction

Classical Automata Theory (AT) is mainly about devices that operate in discrete time. Recent research stimulated the interest to, and the development of, paradigms in which continuous time is involved whether in a pure way or in cooperation with discrete time. This development is in particular evident in the area that covers the following three interrelated trends: automata, logic (arguing about automata) and interaction (composition of automata).

Unfortunately, the area is dominated by a plethora of concepts, terminology and notation, which is not free of ad-hoc and ambiguous decisions that are liable to misjudgments. A wrong formal decision can be misleading in long term even if it turned out to be instrumental in concrete specific cases. It engenders further models and formalisms, and it is not clear where to stop. Hence (quoting J. Hartmanis) the challenge *"to isolate the right concepts, to formulate the right models, and to discard many others, that do not capture the reality we want to understand..."* A comprehensive undertaking of this challenge, covering the three trends mentioned above is a large-scale task-work. Here we confine only to some continuous-time paradigms that are suggested by the current literature on hybrid automata and related control problems. In this literature are often used cumbersome wordings to define "from scratch" intricate notions. For example, here is a (slightly edited) extraction from the definition of the core notion "hybrid automaton" ([VdSS]):

"A hybrid automaton HA is described by a septuple $(L, X, A, W, E, Inv, Act)$ where the symbols have the following meanings:

L is a finite set ... (of) locations,

X is...

A is...

W is...

E is a finite set of edges called transitions (or events). Every edge is defined by a five-tuple $(l, a, Guard_{U'}, Jump_{U'}, l')$, where...

Inv is...

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 4–23, 2001.

Act is a mapping that associates to each location $l \in L$ a set of differential-algebraic equations F_l... (Their) solutions are called the *activities* of the location."

Before attacking such intricate models and tackling the related control issues, it is important to reach a clear understanding and, ultimately, to provide definitions for some basic proviso. Namely,

(i) What are continuous-time (for short, continuous) automata and what is their purposeful classification?

(ii) What are the relevant models of interaction and control for continuous automata, and how do they relate to the original counterparts in classical automata theory?

After this is agreed upon, the next question to be addressed is:

(iii) What are (and what should be) hybrid systems (in particular, "hybrid automata")?

We believe that first, the agenda should build on mainstream, basic Automata Theory and related, well-understood tools. Hence, on the conceptual level there is no reason to involve abstract algebra and/or calculus, say differential equations. For example, instead of the specific *activities* (we call them also flows) and *jumps* that occur in the definition above, one would rely on the following

Definition 1. *1. A jump j on state-space X is a function of type $X \longrightarrow X$; it is total if it is defined everywhere in X.*

2. A flow f on state-space X is a function $f : X \times R^{\geq 0} \longrightarrow X$, that meets the following conditions:

(i) $f(x, 0) = x$; , and (additivity):

$$f(x, t_1) = x' \ \& f(x', t_2) = x'' \ \longrightarrow f(x, t_1 + t_2) = x'',$$

(ii) if $f(x, t)$ is defined, so is $f(x, t')$ for each $0 < t' < t$,

(iii) the flow is total if f(x, t) is defined for all $x \in X, t \in R^{t \geq 0}$; otherwise it is partial. "nil" is the notation for the (polymorphic) trivial flow: $\forall t[nil(x, t) = x]$.

Suppose that the space X is actually the Euclidean R^n. Then, some flows could be specified as solutions of appropriate differential equations, and some jumps, as linear transformations. Clearly, information of this kind may be crucial for those who, for example, compute reachable states of specific continuous systems. Nevertheless, this information is not part of the basic concepts we pursue. These are driven to some degree by mathematical curiosity: how can one lift classical automata theory to deal with continuous time or ascertain that this is impossible. As to potential applications, we content (following an apt expression of C. Rota) with "hygienic prescriptions meant to guard us against potentially unpleasant complications". Hopefully, this will become clearer in the concluding section 8 (Discussion).

A major goal is to highlight the distinctions and the similarities between the discrete and continuous tracks. Hence, the policy of building up definitional

suggestions in an incremental/orthogonal style; this allows to gradually estimate the effect of introducing continuous time.

Eventually, in spite of similar terminology, a continuous-time entity is different from (and more involved than) the corresponding one in discrete time. For example:

(i) In discrete time, the execution of a transition is considered as an instantaneous event; for continuous time the duration aspects become relevant.

(ii) The unit delay is a finite automaton in the discrete case, whereas continuous time forces the delay to memorize an uncountable amount of information.

On the other hand, some phenomena may look as being characteristic for continuous time, merely because their discrete-time counterparts were not explicitly identified previously. Hence, the need to spell out accurately the related issues and to highlight their universal character.

Below is a preliminary outline of what is to come. For many reasons (in particular, because the lack of precise and elegant terminological standards) we are forced to an eclectic, informal style. The numerous and somehow evasive "Observations", spread over the whole exposition, are intended mainly to help comparing the discrete and continuous tracks.

We start with a brief (and not fully conventional) presentation of some discrete stuff, whose continuous-time analogs and/or mutants we would like to understand. Further we build on a generic definition of the automaton concept via a system of axioms **Ax** borrowed from the Control-Theory monograph [S1]. Actually, this system is conceived for both discrete and continuous automata, depending on whether the underlying time-domain $TIME$ is the ordered set Nat of natural numbers or the real line $R^{\geq 0}$.

Continuous automata. Having chosen the real line. one can move toward more specific continuous automata via expedient concretization of **Ax** and keeping to the following principles:

(i) Consistency with **Ax**;

(ii) "Continuity" of automata refers only to their time-domain, i.e. to the real line, equipped with its standard order, metrics and algebraic structure. As to the input, output and state-spaces (alphabets), whether finite or infinite, they are handled as amorphous spaces with discrete topology.

With these reservations, we mention immediately some germane features of continuous automata that are not visible, or have different effects in discrete time. Yet, none of them (nondeterminism, completeness and duration independence) is considered explicitly in the literature.

On the other hand, on the intuitive level there is a consensus in the community about restrictions to be imposed on signals manipulated by continuous automata. Namely,

(i) "Realistic" input signals, whether with finite or with infinite duration, should enjoy the property known as *finite variability* (FV), i.e avoidance of Zeno-anomaly.

(ii) Under the influence of such input signals, a continuous automaton should either evolve continuously for some duration of time (a flow phase), or should be abruptly reset to a new value (a jump hit) from which subsequent continuous evolution occurs.

At a lower degree of consensus is the following feature:

(iii) (Flow robustness) The behavior of the automaton is not influenced by "sparse" deviations which do not affect the jump hits.

We aim at a formalization of these intuitions in a way that is consistent with **Ax**.

Input/output. So far, about outputless automata; we call them simply "automata". The term "transducer" is preserved for an automaton M equipped with a *readout* $\Psi : X \times U \longrightarrow Y$, where Y is a *finite* (!) output-space. As in the discrete case, a transducer T computes (implements) an input/output operator F which is retrospective, i.e. the output-value at time t depends only on the values inputted not later than t. If the readout Φ depends only on the current state of M it is called *measurement*. In this case, the i/o-operator F is strongly retrospective, i.e. the output at time t depends only on the values inputted before t. For a given retrospective operator F there may be different implementations. In most of the issues concerning circuits and/or control the essence is in F and not in its concrete implementation. This remark refers to both discrete and continuous time.

In discrete time, strong retrospection and retrospection reflect sharply different causal dependencies of output from input. Under appropriate (Baire) metrics, retrospection of an operator F implies its continuity. On the other hand, strong retrospection implies the Lipshits property i.e. implies that F is a contracting map. This paves the way to using strongly retrospective operators in fixed point techniques needed to tackle feedback in circuits and in control. Yet, in continuous time causality is more subtle and is not exhausted by the dilemma "retrospection vs. strong retrospection"; hence, the need to define and analyze more subtle properties of retrospective operators: predictivity, weak delaying, bounded output variability etc.

Nondeterminism. In both discrete and continuous tracks it is handled in the $\exists$-setup. Namely, a nondeterministic object Ob (automaton, transducer, operator) is represented (implemented) by a deterministic object Ob' with appropriately hidden (shady) input.

Interaction. A comprehensive taxonomy of interaction relies on the following dichotomies ([T4]):

1. Synchrony versus asynchrony.
2. Disjoint memory versus shared memory.

Here, we are interested mainly in synchrony of automata (respectively, transducers) with disjoint memory ("disjoint state-spaces").

The crucial point about systems of interacting transducers is that, unlike systems of interacting automata, they are subjected to a specific *feedback discipline*; we call such systems in brief - circuits. Originally, they were identified as faithful,

idealized models of *digital (!)* computer circuits. This is the *hardware view* on circuits. In [BW], Burks and Wright coined the name Logical Net for circuits composed of logical gates and delays working in discrete time. Circuits should be feedback-reliable, i.e. intuitively, the propagation of signals along closed cycles should be causally faithful. In continuous time, feedback-reliability is more subtle because the (non existing in discrete time) danger of *Zeno*-anomaly.

A control view on circuits. Here the focus is on the interaction between two main components, the plant and the controller. In discrete time, synthesis of control circuits amounts to finding *appropriate* controllers for given plants. In continuous time, synthesis includes also the design of a third component, namely the *interface*. As mentioned earlier, there is no commitment to specific techniques from differential equations and/or numerical analysis; however, one still has to face non trivial feature of the circuit's components.

2 Discrete Time

From now on, X, U, Y are typical notations for state, input and output alphabets (spaces); U and Y are finite. Typically, u is an element of U, $\tilde{u}$ a path with values in U and, finally, $\tilde{U}$ the set of such paths. Similarly for other alphabets. A deterministic discrete automaton M is usually identified with a map $nextstate : X \times U \longrightarrow X$. When applied to a state x and an input sequence $\tilde{u} = \tilde{u}(1)...\tilde{u}(n)$, the automaton returns a state-sequence $\tilde{x}$ of length $n+1$. Namely,

$$\tilde{x}(1) = x;\ \tilde{x}(i+1) = nextstate(\tilde{x}(i), \tilde{u}(i))$$

Correspondingly, with *nextstate* are associated:

1. The *terminal transition map* $\Psi : X \times \tilde{U} \longrightarrow X$ which returns the value $\tilde{x}(n+1)$;
2. The *full transition map* $\tilde{\Psi} : X \times \tilde{U} \longrightarrow \tilde{X}$ which returns the sequence $\tilde{x}(1)...\tilde{x}(n)$. Assume that an initial state is fixed. Then, $\tilde{\Psi}$ induces a strongly retrospective input/state (i/s) behavior.

Observation 1. Nextstate, terminal and full transition maps can be uniquely restored from each other. Full transition makes sense for infinite input sequences.

Nondeterminism. The terminal and full transition-maps $\Psi_D, \tilde{\Psi}_D$ of a deterministic automaton D represent the terminal and full *transition-relations* R_D, $\tilde{R}_D$ of a $\exists$-automaton. Assume that the input-alphabet of D is $U \times V$, and, correspondingly, its input paths are $< \tilde{u}, \tilde{v} >$. Then, the terminal transition-relation R_D^V and the full transition-relation $\tilde{R}_D^V$ which are implemented by D via the hiding (projecting out) of V, are as follows:

$$R_D^V(x, \tilde{u}, x') \text{ iff } \exists \tilde{v}[\Psi_D(x, < \tilde{u}, \tilde{v} >) = x']$$

$$\tilde{R}_D^V(x, \tilde{u}, \tilde{x}) \text{ iff } \exists \tilde{v}[\tilde{\Psi}_D(x, < \tilde{u}, \tilde{v} >) = \tilde{x}]$$

Since the ∃-implementation is not unique the following observation about the implementation is instructive:

Observation 2. (Robustness of ∃-implementation) If $R^V_D = R^{V'}_{D'}$ then $\tilde{R}^V_D = \tilde{R}^{V'}_{D'}$

Circuits. The analysis in depth of feedback and causality issues is an essential aspect of circuit theory. Most regrettably, this point is often missing in the literature. We remind below some details ([BW], [KT]:

At the most appropriate level of abstraction, a circuit C of k interacting transducers T_i communicating through their inputs and outputs, is specified by a recursive system Eq of k functional equations Eq_i with the format $x_i = f_i(y_1, ..., y_{n_i})$; $i = 1, ..., k$. Each equation describes one of C's components, and f_i is a retrospective operator, namely the i/o-behavior of the transducer T_i. All the variables occurring in Eq are typed; those which occur only on the right hand side of an equation (say $v_1, ..., v_m$) are declared as input variables of Eq, the others (say $w_1, ..., w_n$) - as its output variables. The conjunction $\bigwedge Eq_i$ defines a $(m+n)$-dimensional relation $R(v_1, ..., v_m; w_1, ..., w_n)$ which, by chance, may happen to be the graph of a total operator $< w_1, ..., w_n >= F(v_1, ..., v_m)$. Say that F is a (functional) solution of Eq. For each f_i, let f_i^δ denotes its δ-truncation, i.e. the restriction to paths with duration $\leq \delta$; respectively, consider the δ-truncations Eq^δ of the system Eq and their solutions. Only "good" solutions of Eq (if they exist) are relevant for the input/output discipline.

Definition 2. *The system Eq is well behaved if it has a unique i/o-solution F with infinite duration; yet, in general, F is not necessarily retrospective. Eq is sound if F is retrospective, and, moreover: for each finite δ, the truncated system Eq^δ has an unique δ-restricted solution, which is also retrospective. A circuit is sound (synonymously, feedback-reliable) if the corresponding system Eq is sound.*

Observation 3. The system Eq is sound if it satisfies the conditions:

(i) no two equations have the same left side,

(ii) each cycle in Eq passes through a strongly retrospective operator.

Circuit C^* with nondeterministic components T_i^*. Syntactically, instead of equations Eq_i with format "variable = term", appear inclusions with format"variable ∈ term". Actually,, according to the ∃-approach, C^* is handled as a circuit C in which the T_i^* are replaced by corresponding deterministic implementations T_i; it is required that the shady input of T_i does not appear in any other equation. Hence, in C, beyond the input variables inherited from C^*, all the shady variables have also the status of input variables. What it remains to do, is to check the soundness of C and to hide the relevant shady inputs.

Observation 4. (Robustness). The semantics of C^* does not depend on the choice of the implementations T_i for the given T_i^*.

Constraining and Control. Given an automaton M one may be interested only in those "good" trajectories or input-paths that meet some requirement $\mathcal{F}$. Let us look to *safety requirements*. A natural way to their formalization is via an appropriate *enabling relation* E, where $E(x, a)$ means " in the state x of M is allowed (not prohibited) the input a. The relation E may be identified

with the function $menu_E(x)$ which associates with x the *finite* set of those a that are enabled at x. Alternatively, one may use *admissibility regions*; namely, $adm_E(u_i)$ is the set of states in which the given a_i is enabled. Correspondingly, a trajectory $< \tilde{u}, \tilde{x} >$ of M is E-safe iff it does not violate E at any time-instant t; in this case, the input-path $\tilde{u}$ is also said to be E-safe. Saying it otherwise, the pair $< M, E >$ defines a subautomaton M' of M, whose trajectories are the E-safe trajectories of M.

Yet another universal way to constrain an automaton M is to consider the synchronization $N = M \times D$ with a companion automaton ("inhibitor) D .

A combination of both ideas is via *synchronized enabling*. It is assumed that M, D have the same input-alphabet U and disjoint state-spaces X, S; on the other side, the enabling E is chosen for N and, hence, induces admissibility regions in the state-space $X \times D$. The projection of $< N, E >$ into X is the intended constraining of M.

According to another popular constraining mechanism (by *simple guards*), the admission or prohibition of a transition at the current state may depend also on the value inputted at the preceding time-instant. This means that in addition to E one considers also a *guard relation* G. Namely, $G(x, u_i, u_j)$ means: "u_j is not prohibited at state x if it was preceded by u_i. Hence also the use of guard regions $g(u_i, u_j)$ in addition to admissibility regions $adm(u_i)$.

Observation 5. Synchronized enabling is more powerful than enabling + simple guards. In particular, finite-state inhibitors cover the effect of "regular" guards, which take into account more than the immediately preceding input values.

The constraining of a transducer P reflects rather the control view on circuits. Here, the constraining of P is via synchronization with an appropriate companion-transducer ("controller") Con. It is required also that P together with Con make up a *feedback reliable* circuit C. In view of the specific, simple structure of the intended C, feedback-reliability issues are simpler than in the general "hardware" perception. The controller problem is understood as follows: given a plant P and requirement $\mathcal{F}$, find (synthesize) the appropriate controller Con or ascertain that this is impossible.

Observation 6. The celebrated synthesis problem in the theory of finite automata and its solution by Büchi can be reformulated in these terms. Note that it does not refer specifically to safe requirements; moreover, it can manage with circuits that are acyclic and, hence, are trivially feedback-reliable. However, the use of suitable cycles may be justified by additional performance criteria.

3 The System Ax: First Refinements

In continuous time, the paths manipulated by M are *signals* i.e. functions from intervals $[t, t+\delta) \subseteq R^{\geq 0}$ (here, δ may happen to be ∞) into the appropriate space (alphabet). Hence, for continuous M, *nextstate*-map does not make sense, and it is natural to start directly with a *terminal transition map* $\Psi : X \times \tilde{U} \to X$.

Note that non-empty input-signals have some positive duration δ. The intended semantics of a terminal transition is: if state x occurs at time t, then $\tilde{u}$

with life-time $[t, t+\delta)$ produces the state x' at time $t+\delta$. The empty signal ϵ is sometimes used for the sake of algebraic aesthetics. Below, $\tilde{u}_1, \tilde{u}_2 \in \tilde{U}$ are finite paths, $\tilde{u}_1 \cdot \tilde{u}_2$ designates their concatenation. Here is the system **Ax**:

- (Ax1) *Semi-group.*

$$\Psi(x, \epsilon) = x;\ [\Psi(x, \tilde{u}_1) = x' \ \&\ \Psi(x', \tilde{u}_2) = x''] \rightarrow \Psi(x, \tilde{u}_1.\tilde{u}_2) = x''$$

- (Ax2) *Restriction (Density).* Assume that $\Psi(x, \tilde{u}) = x''$. If $\tilde{u}$ is the concatenation $\tilde{u}_1 \cdot \tilde{u}_2$ then there exists x' such that $\Psi(x, \tilde{u}_1) = x'$ & $\Psi(x', \tilde{u}_2) = x''$.
- (Ax3) *Non-triviality.* For each state $x \in X$ there is a nonempty input-path admissible at x.

Terminological/notational remarks. Wrt a signal $\tilde{z}$ with values in some Z, continuity, right-continuity, right limit etc. refer to the discrete topology. For example, $\tilde{z}(t+0) = a$ would mean that $\tilde{z}$ is defined and has the constant value a in some interval $(t, t+\delta)$. Say that an ordered set of time-instances is *sparse* iff it is finite or an increasing sequence $t_1 < t_2 < ... < t_i < ...$ with $t_i \longrightarrow \infty$. "Almost" (wrt time) means "except a sparse set of time instants".

Observation 7. The following two definitions are germane for continuous automata but do not make sense for discrete automata.

Definition 3. *The automaton M is complete iff it satisfies the following condition; Consider a state $x \in X$ and an input path $\tilde{u}$; If arbitrary proper prefix $\tilde{u}'$ of $\tilde{u}$ is admissible at x, i.e. $\Psi(x, \tilde{u}')$ is defined, so is $\tilde{u}$.*

It is important to distinguish metrical aspects of theory, which deal with the distances between time-instants, from duration-independent aspects that reflect only the order of time instances.

A *stretching* of the time-axis $R^{\geq 0}$ is an arbitrary 1-1 monotonic mapping $\rho : R^{\geq 0} \rightarrow R^{\geq 0}$. A path $\tilde{w}$ is the $\rho-$stretching of the path $\tilde{v}$ if

$$\forall t(\tilde{w}(t) = \tilde{v}(\rho(t)).$$

Definition 4. *The automaton M is duration-independent if each stretching ρ of an input path causes the $\rho-$stretching of the corresponding state-path.*

$\exists$-nondeterminism. The formal definitions of the transition-relations are exactly as in the discrete case. However, in continuous time it may happen that two deterministic automata D, D' implement the same terminal transition-relation but different full transition-relations. Hence,

Observation 8. The robustness mentioned earlier for discrete-time automata fails, in general, for continuous automata. The conclusion: build on full transition-maps.

Note that some popular versions of nondeterminism, in particular that of *nondeterministic delay* ([MP1]), can be easily reformulated in (*desugared to)* the $\exists$-track, even though the usual definitions do not appear so.

4 Realistic Features of Signals

Say that $\tilde{z}$ is an elementary path with duration $\delta \in R^{>0} \bigcup \{\infty\}$ iff for some $a, b \in Z$ there holds: $\tilde{z}(0) = a$; $\tilde{z}(\tau) = b$ for $0 < \tau < \delta$. The corresponding notation is $\tilde{z} =< a \bullet b, \delta >$. Finite variability ($FV$) of a path $\tilde{u}$ means that it can be presented as a concatenation of elementary path-components; moreover, the number of these components is finite (respectively, infinite) if the duration of $\tilde{u}$ is finite (respectively, infinite). M is a FV-automaton iff all its input-paths have FV. Hence, up to semigroup closure, for FV-automata, it suffices to define the terminal transition function Ψ only for elementary input-paths. The self-explanatory notation is $\Psi(x, u \bullet u', \delta)$.

In order to formalize the intuitive expectation about flows and jumps we assume first that:

1. The input alphabet U is the disjoint sum of U^J (the jump-sub-alphabet with generic elements j_l) and U^F (the flow-sub-alphabet with generic elements f_r).
2. On each input-path $\tilde{u}$, the set of jump-instances is sparse.

Since we consider FV-automata, we have only to characterize elementary transitions of two kinds:

$$\Psi(x, f_i \bullet f_s, \delta)\ (pure\ flow) \qquad (*)$$

$$\Psi(x, j_i \bullet f_s, \delta)\ (flow\ after\ jump) \qquad (**)$$

This is done (and checked to be consistent with **Ax**) as follows:

3. With each $f \in U^F$ is associated a flow $\|f\|$ and, *independently of* f_i *(!)*

$$\Psi(x, f_i \bullet f_s, \delta) \ =_{def} \ \|f_s\|(x, \delta)$$

.

4. With each j is associated a jump $\|j\|$ and

$$\Psi(x, j \bullet f, \delta) =_{def} f(\|j\|(x), \delta)$$

.

Say that M is a *jump/flow (j/f)*-automaton if it can be specified as above. M is a flow automaton if U^J is empty. If all the paths of M are right continuous it is said to be a right-continuous automaton; clearly, this implies that M is a flow automaton. M is a jump automaton if its only flow is nil.

Two input-paths of M are flow-similar if the set of time-instants where they differ is sparse, and at each such instant both have flow-values.

Observation 9. (Flow robustness) A jump/flow-automaton M does not distinguish input-paths which are flow-similar.

Duration Independence under Finite Variability. In this case an automaton is *duration independent* if whenever $\Psi(q_1, a \bullet b, \delta) = q_2$ holds for some

duration δ, it holds for arbitrary δ. Hence, the δ argument may be omitted. Clearly, a jump automaton is duration independent. The following is not trivial:

Proposition 1. ([R]) *Every FV-automaton M with finite state-space is duration independent.*

It is worth noting that, even though this result is about FV-automata, the proof in [R] leaves the world of finite variability and deals with arbitrary paths.

Example (trivial). Modeling a finite-state discrete automaton M with a duration-independent automaton (actually with a jump automaton M^J).

Consider M with state-space X, input-alphabet $A = \{a_1, ..., a_k\}$ and transition-map $nextstate_M$. First, extend M to M' with input alphabet $A' =_{def} A \bigcup \{nil\}$ and with additional (to $nextstate_M$) transitions $nextstate_{M'}(x, nil) = x$. Now, the state and input-spaces of M^J are the same as for M'. Its elementary transitions mimic those of M' in the most natural way: for each $u \in A'$

$$\Psi^J(x, u \bullet nil) = nextstate_{M'}(x, u) \tag{1}$$

We confine to this trivial example and mention, in a somehow vague form, a general fact to be used in the sequel (it holds also for transducers).

Observation 10. In a strong sense, duration-independent automata and transducers model (and a modeled by) discrete automata and transducers.

5 Circuits

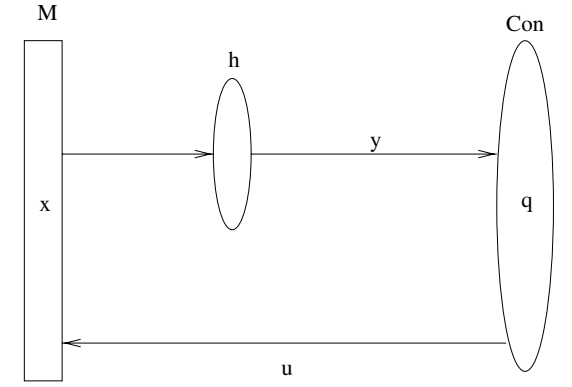
Observation 11. The soundness criterion for discrete-time (see Observation 3) fails for continuous time. The reason: strong retrospection does not support the needed fixed-point techniques.

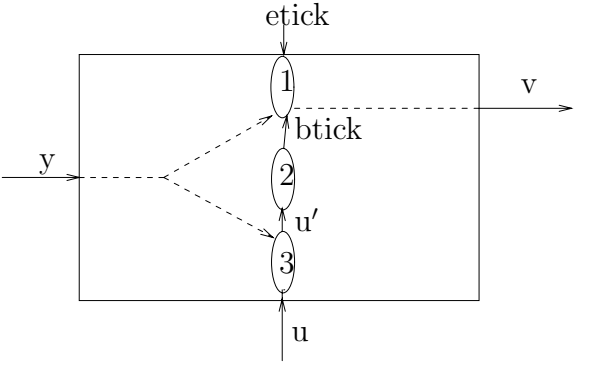
There are sufficient conditions that use additional properties of retrospective operators like: predictivity, bounded variability of output signals etc. ([PRT, T1]). Respectively, there are also some specific "anti-anomaly" continuous-time primitives that differ from delays, but mimic somehow their behavior. Circuits that make use of such primitives can be considered to some degree as models of continuous-time hardware ([PRT]).

Observation 12. In continuous time, circuits over $\exists$-nondeterministic components, can be handled according to the discrete-time scenario modulo two precautions. First, see Observation 11 and the accompanying comments. Second, provide faithful $\exists$-implementations for the nondeterministic components.

6 Constraining Continuous Automata

This is a straightforward adaptation of the discrete-time pattern (section 1). Namely, the format of the enabling E is $E(x, a \bullet b)$, understood as "in the state





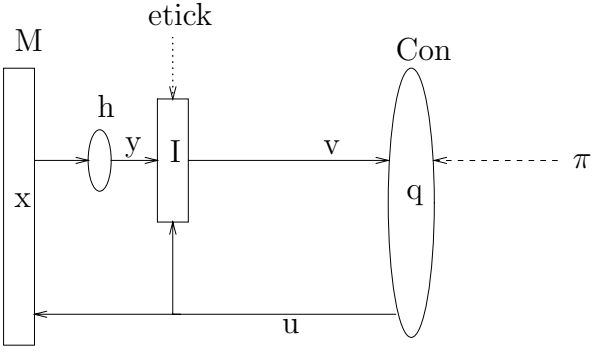


Fig. 1.

x of M are allowed (not prohibited) elementary input-paths $< x, a \bullet b, \delta >$". Further, one can consider the associated $menu_E(x)$, admissibility regions $adm_E(a \bullet b)$, E-safe trajectories and E-safe subautomaton M' of M.

The definition of synchronized enabling is preserved, but note that the inhibitor D is assumed to be a finite-state, hence duration-independent automaton (see Observation 10).

In order to handle guards, one considers, in addition to the enabling relation E, also a guard-relation G, with format $G(x, u_i, a \bullet b)$. Here, u_i is (in a very precise sense) the flow that precedes a potential elementary input path $< a \bullet b, \delta >$. Correspondingly, are considered guard-regions $g(u_i, a \bullet b)$. Finally,

Observation 13. With the reservations above, the former Observation 5 is valid for continuous time as well.

7 Continuous-Time Control

In continuous time, a control circuit C may contain, in addition to the Plant P and Controller Con, also an interface I. Since P, Con, I are not subjected to further decomposition, feedback reliability issues are simpler than for hardware-oriented circuits.

We confine below to control circuits with *deterministic* (!) plants, that mimic *Sample-and-Hold (SH)* architectures from Control Theory. Namely, the values y outputted by the plant P are available to Con only at some *sampling instants* $t_0 = 0 < t_1 < t_2 < ....$ The controller is updated at each t_i and, on the basis of the information available at t_i, it computes the output for the life-time $[t_i, t_{i+1})$. In Control Theory, it is often assumed that $|t_{i+1} - t_i|$ is constant; in such cases the similarity with discrete time is quite evident. However, in general one has to build on appropriately scheduled sampling-instants.

A simple kind of control is *selection*. Assume a total P and some provisions $\mathcal{F}$ imposed on its input paths. The task is to find appropriate deterministic Con and I such that the circuit C over $< P, Con, I >$ restricts P to a *unique* (!) input-path which meets $\mathcal{F}$, or ascertain that this is impossible. We are going to illustrate this wrt *safe* selection. In this case it is assumed that the plant P is represented explicitly as an automaton M equipped with a measurement h, and that the requirement $\mathcal{F}$ is formulated in terms of an *enabling relation* E for M, exactly as in the previous sections 1 and 5. Correspondingly are understood the notions E-safe trajectories, and "subautomaton M' of M" defined by the pair $< M, E >$

Intuitively, a positive solution presumes that the current output of P can serve as a basis for feedback control, i.e. it is enough informative and it grants to the potential controller the chance to properly react in real time. Hence, the assumption:

$$h(x) = menu_E x \qquad (h)$$

Under this condition, the control circuit is required to select a unique trajectory of M which is E-safe. Actually, the trajectory will have the additional property of *persistence*. Namely, no change of the current flow occurs as long as it is

enabled. Clearly, this means that the circuit C should possess some kind of "persistence ability"; namely it should be able to properly register and react to border-crossings of admissibility regions.

An appropriate *Sample-and-Hold* architecture is depicted in the lower part of Fig. 1. Here, I is the interface, whereas P is represented via M and h.

The controller Con is a finite-state (hence, a duration-independent) transducer with input-alphabet $V = Y \bigcup \{nil\}$ and output-alphabet U. The cardinality of the state-space Q may vary depending on the intended selection strategy. Con inputs values $\in Y$ at the sampling instants and inputs the value nil otherwise.

Observation 14. Assume that the subautomaton M' satisfies two conditions: finite variability and completeness. Then the circuit selects an input-path $\tilde{u}$ with finite-variability which is both safe and persistent.

Here are some details that characterize the *persistence ability* of the circuit and the importance of the *completeness (!)* assumption about M'. It turns out that the interface I is responsible for the appropriate time-management of the sampling instants. As shown in the middle part of Fig.1, it consists of three components: (Note the error in the picture: the dashed line directed to 3 should be directed to 2).

Component 1. This is the transducer $Sieve$ with input y and, in general, with two more "ticking" inputs $etick$ and $btick$. In the deterministic architecture one manages only with $btick$ and

$$v(t) = y(t) \text{ iff } btick(t) = tick \ ; \quad \text{otherwise} \quad v(t) = nil$$

At isolated time-instants $Sieve$ outputs to Con the values $\in Y$ to be sampled.

Component 3 implements a strongly retrospective operator; namely, $\tilde{u}'(t) = \tilde{u}(t-0)$

Component 2. This is the *Boundary Detector* B which is at the heart of the "persistence ability". The role of B is to detect the next time-instant t (if any) when the current flow u_i is not enabled anymore, i.e. when $u_i \notin menu_E(x(t))$; in other words, at t is hit the border of the region $adm(u_i)$. According to the expected persistence ability, only at such instant may occur a fresh sampling. B implements the operator:

$$btick(t) = tick \text{ iff } \tilde{u}'(t) \notin (y(t)); \text{ otherwise } = nil$$

(Remind that $y(t) = h(x(t)) = menu(x(t))$. Hence, unlike the other two components, the Boundary Detector B is tailored especially to the concrete plant, notably to $menu_E(x)$, which is just the output y. See that all this works OK due to the completeness assumption!

8 Hybrids

According to Webster's Dictionary a hybrid is "something heterogeneous (consisting of dissimilar ingredients or constituents) in origin or composition".

To be more precise we expect that:

(i) The heterogeneous constituents are two agents: the continuous one L and the discrete (may be even, finite) one D.

(ii) In order to constitute a (hybrid) system, the two should be subjected to an appropriate composition.

As suggested by our previous exposition it seems that:

a) L should be, in general, a jump/flow automaton or a transducer with an underlying jump/flow automaton; sometimes one may confine to flow-automata.

b) One should give up the tempting option to draw in discrete automata. Instead, in accordance with Observation 10, one can use their faithful duration independent models.

c) For automata, the main composition rule is synchronization with disjoint memory. For transducers, because the relevance of feedback, it is natural to use circuits. Actually, one more synchronization-like composition (but with fully shared memory) seems to be appropriate. We call it "blend"; the definition appears a bit later.

Proceeding from the assumptions above, we mention now three kinds of hybrid systems, to which we refer as H1, H2, H3. Note that the first two (H1 and H2) refer to automata (hence, no involvement of input/output), whereas in H3 the concern is about transducers (hence, also, the relevance of circuits and feedback). On the other hand, H2 and H3 have natural counterparts in discrete time, whereas H1 is based exclusively on continuous-time peculiarities. Here are all the three:

H3 is a feedback-reliable control circuit.

H2 (see the middle of Fig. 2) is the synchronized enabling (discrete version in sections 1, the continuous one in section 5). There is an extensive literature dedicated to infinite-state generators of "safe" trajectories. Most of them converge to the so called "Hybrid Automata" (HA), alleged to be a fundamental paradigm which combines continuous and discrete dynamics. This model is, in fact, a particular, disguised case of the H2-concept; unfortunately, the underlying composition is not spelled out explicitly in ([ACHP, ABD*, VdSS]). Here are some details:

Picture $\mathcal{H}$ in Fig. 2 displays a widely accepted way to explain "Hybrid-Automata" as specific transition systems with labeling for edges (here, g_{ij}) and vertices (adm_i and f_i). The operational semantics characterizes the corresponding safe trajectories, as runs which respect the "invariants" adm_i and the guards g_{ij}. The activities f_i are usually specified by differential equations.

The corresponding H2-model is partially explained in the respective part of Fig. 2. Here, the inhibitor D is a delay with inputs u_1, u_2, (the same as of M) and states s_1, s_2. On the other side, M is a flow automaton with flows borrowed from $\mathcal{H}$. Finally, the admissibility regions of N are as follows:

$$(s_i, x) \in adm_N(u_j) =_{def} \quad x \in g(u_i, u_j) \quad \wedge \quad x \in adm(u_j)$$

H1 is the composition (tentatively called here "blend") which applies to a pair $< M_1, M_2 >$, where M_1 is a jump-automaton and M_2 is a right-continuous automaton (a particular case of flow automaton). The operation produces a jump/flow automaton M; all the three have the same state-space X. Intuitively, M inputs in parallel a jump-path of M_1 and a right-continuous path of M_2, as suggested by part H1 in Fig.2, and performs their mixture. Let U_1, U_2, U be the corresponding input alphabets. Here $U_1 =_{def} \{j_1, ..., j_l, ..., j_m; nil\}$ and $U_2 =_{def} \{f_1, ...f_r, ..., f_n\}$. Then, $U =_{def} U_1 \times U_2$, i.e it consists of pairs $(j_l, f_r) \in U^J$ (these are the jumps of M) and of pairs $(nil, f_r) \in U^F$ (these are the flows of M).

Definition 5. *(*†−composition)

- *A path $\tilde{z}$ is admissible in M iff its projections are admissible in the respective components. Hence,*
- *Elementary input-paths of M have (up to duration) one of the formats:*

$$\alpha =_{def} (j_l, f_s) \bullet (nil, f_s); \quad \beta =_{def} (nil, f_s) \bullet (nil, f_s) \tag{2}$$

- *Correspondingly, the transition map Ψ_M is defined as follows:*

$$\Psi_M(x, \alpha, \delta) = f_s[j_l(x), \delta)]; \quad \Psi_M(x, \beta, \delta) = f_s(x, \delta)) \tag{3}$$

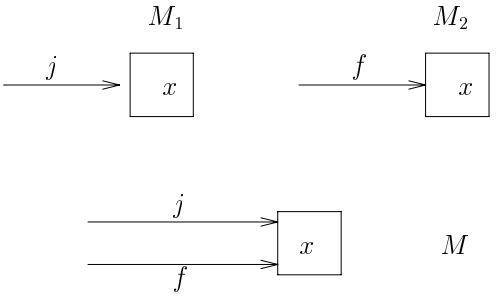
9 Discussion

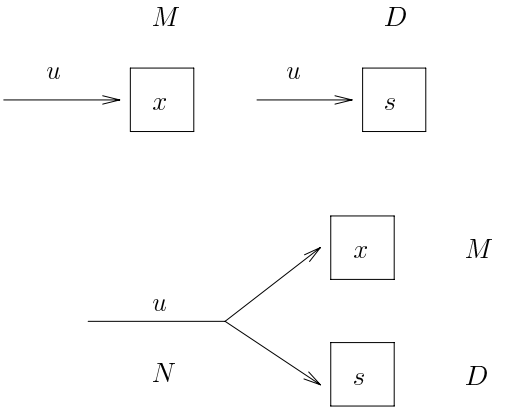
About the Sample-and-Hold architecture. Section 6 can be generalized in two orthogonal directions:

(i) Instead of selection consider uniformization. This means that beyond the plant's input u fed by the controller, the plant has an additional input e fed by the environment. On the other hand, the controller is required to guarantee correct selections in the face of all the behaviors of the environment. This generalization can be handled via a routine adaptation of selection.

(ii) Instead of deterministic selection consider nondeterministic ones. This needs a recasting of the deterministic architecture (section 6), in which all three components were deterministic. In a nondeterministic architecture, the plant is still deterministic; the ∃-nondeterminism is incarnate in the controller *Con* (see the shady input π in the lower Fig, 1) and in the interface I (the shady input *etick*). There is a clear difference between:

1. Nondeterministic detection of sampling-instants. In addition to the internal mechanism of boundary detection (which is deterministic and relies on the *completeness* assumption) here are relevant the timing ticks supplied by the external shady input *etick*.
2. Nondeterministic choice of the current flow (or flow after jump). This choice has to be done in accordance with the current available *menu*; it is fully on the responsibility of the controller.


M_1
M_2
j
x
f
x
H1
j
x
f
M


M
D
u
x
u
s
H2
x
M
u
N
s
D

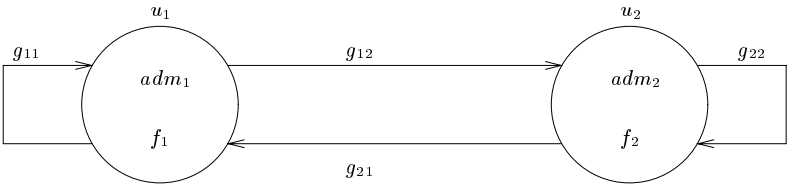

u_1
u_2
g_{11}
g_{12}
g_{22}
adm_1
adm_2
f_1
f_2
g_{21}
$\mathcal{H}$

Fig. 2.

Synthesis of controllers in [ABD*] (it may be consulted for other references). Here are the main features:

(i) Beyond safety, is handled also the important liveness property called viability (nonblocking).

(ii) There is no demand that control should be reached via feedback with an explicitly produced companion-controller. In addition to these basic distinctions we observe also:

(iii) There is no reference in [ABD*] to *completeness* of automata and to the possible impact of this property on the subject.

And now, more details.

Viability requires that each finite trajectory of the safely constrained automaton should be extendible at infinitum to a safe trajectory. Since, in general, this is not the case, the following task arises: reinforce the original safety constraints into new ones such that the automaton fits viability wrt them. The authors develop a procedure (Algorithm 1) intended to produce the less restrictive reinforcement of this kind. The constrained automaton produced by the procedure above is called by the authors "controller", whereas the procedure itself is identified with the synthesis of the controller.

Further (quotation from [ABD*] follows): "... The termination of the procedure, however, cannot be guaranteed... Moreover, the implementation is very complicated... Some aspects of the technique take advantage of special properties of linear systems. Given this state-of-affairs we must resort to the classical solution of continuous mathematicians, numerical approximation."

But, beyond these troubles, one more fact is worth attention. One can show that the procedure may produce an *incomplete* automaton even if the original one *is complete*. This may happen to be relevant for the next point.

Referring to the determinization of their controller, [ABD*] claim (without proof): "This can be fixed later ... and the feed-back map will become a function..." Indeed, this is evident wrt the nondeterministic choice of the current flow. Moreover, *under the assumption that the resulting "controller" is complete*, the techniques of boundary crossings would allow also to fix the nondeterminism of time-detection. However, as we just mentioned, this argument cannot be used. Actually, we doubt if this can be cured in some other way.

About Hybrids. Nice names may happen to precede formal concepts and, eventually, to engender different concepts. Hence, the controversial question: which of them captures better the intended reality we want to understand and deserves to preempt this name. By now there are various versions of "Hybrid System" and the related Control problems.

The conceptual/notational approach in [H] (which may be consulted for further references), focuses on "Hybrid Automata". It differs from that advocated in this paper as follows:

(i) No consideration of operators/transducers, feedback reliability.

(ii) Use of instantaneous transitions. Remember that, according to **Ax**, the only instantaneous transition is associated with ϵ.

(iii) Multiform time. This amounts to breaking the time-axis $R^{\geq 0}$ into a sequence of closed intervals, which may reduce to single points. Accordingly, legitimacy is given to "signals" (called in [MP3] "functions which are not really functions") that may have different values at the "same" time-instant. Clearly, this is inconsistent with **Ax** , whose signals are genuine functions.

(iv) Inclusion of asynchrony in the basic model of Hybrid Automata.

(v) No explicit representation of the hybrid as a pair of interacting components.

The last shortcoming is criticized and remedied in [OD] for timed automata (a particular case of hybrid automata).

Quotation. "...real-time system can be decomposed into an untimed system communicating with suitable timers. Both synchronous and asynchronous communications are considered... At first sight it seems that the decomposition...is already solved by the model of timed automata... However, the main difference... is that in the timed automata model the clock operations are indivisible coupled with the transitions whereas here we present a clear separation of untimed system and timers with explicit communication between them".

In [DN] Hybrid Control Systems are explicitly structured as interacting plants, controllers and interfaces. However, feedback-reliability issues are not considered. In [DN] there is no explicit reference to completeness; on the other side, there is a clear, inspiring presentation of the border-crossing mechanism, which suggests the importance of this property.

Circuits in [MP1]. In terms comparable with section 4, it seems that the intention is to circuits (synchronous !) of nondeterministic retrospective operators (the authors use the name "Asynchronous Circuits"). In this case, the subject could be handled after the scenario from Observation 12; the result would be that the circuits under consideration are indeed feedback reliable. However, being absorbed by specific applications, the authors ignore this crucial aspect of circuitry. What they call "solution" of the relevant system $Eq = \{Eq_i\}$ is, actually, the relation $R = \bigwedge Eq_i$ (see section 1) irrespectively of the question if Eq is sound. This deviation from the "circuit"-philosophy is visible already for discrete time. The full scenario would require also to present an $\exists$-implementation for the nondeterministic continuous delay considered in [MP1] .

Control Theory. Here, Continuous Automata (Dynamical Systems) are specified by differential equations. But note that models with discrete time (albeit with continuous data) are also considered ([S2])

In ([A,S2]) hybrids are treated as circuits of appropriate transducers: plants, interfaces, controllers. In [A] is used a very non-trivial interface; the invention of the interface and the check that the circuit is feedback reliable are beyond the concepts and techniques of classical automata-theory.

In [A,S2] the controller is implemented as a timed automaton. According to our definitions, this might be interpreted as follows: in addition to the main Plant and Interface, one uses some auxiliary primitive plants like Timers etc. From this perspective, the controller is again a finite (and, hence, a duration-independent) automaton.

Acknowledgement. Oded Maler was very cooperative and helpful in providing information on the subject of Hybrids.

References

[AD] Alur, R., Dill, D.: Automata for modeling real-time systems. Proceedings of ICALP90, LNCS 443 (1990) pp. 332–335.

[AFH] Alur, R., Fix, L., Henzinger, T.: A deterministic class of timed automata. Proc. CAV'94, LNCS 818, pp. 1-13.

[AH] Alur, R., Henzinger, T.: Logics and models for real time: theory in practice. LNCS 600 (1992) pp. 74-106.

[ACHP] Alur, A., Courcoubetis, C., Henzinger, T., Pei-Sin, Ho.: Hybrid automata: approach to the specification and verification of hybrid systems. LNCS 736 (1993) pp. 209-229.

[A] Artstein, Z.: Examples of stabilization with hybrid feedback. LNCS 1066 (1996) pp. 173-185.

[AMP] Asarin, E., Maler, O., Pnueli, A.: Symbolic controller synthesis for discrete and timed systems, in: Hybrid Systems II, LNCS 999, Springer-Verlag, 1995.

[ABD*] Asarin, E., Bournez, O., Dang, Th., Maler, O., Pnueli, A.: Effective synthesis of switching controllers for linear systems, Proceedings of the IEEE, vol. 88, No 7, July 2000.

[AMPS] Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata, Proc. IFAC Symp. on System Structure and Control, pp. 469-474, Elsevier, 1998.

[B] Berry, G.: The constructive semantics of Pure Esterel, Technical Report (Draft Version 1.2), April 1996.

[Br] Broy, M.: Semantics of finite and infinite networks of concurrent communicating agents. Distributed Computing 2 (1987) 13-31.

[BW] Burks, A.W., Wright, J.B.: Theory of logical nets. Proceedings of the I.R.E., 1953.

[DN] Davoren, J.M., Nerode, A.: Logics for hybrid systems, Proceedings of the IEEE, vol. 88, No 7, July 2000.

[H] Henzinger, T.: The theory of hybrid automata. Proceedings of the IEEE (1996) 278–292.

[HR1] Hirshfeld, Y., Rabinovich, A.: A framework for decidable metric logics, Proceedings of the ICALP'99 colloquium, Editors J. Wiedermann, P. van Emde Boas, M. Nielsen, LNCS 1644, Springer-Verlag, 1999, pages 422-432.

[HR2] Hirshfeld, Y., Rabinovich, A.: Quantitative temporal logic, Proceedings of the CSL colloquium, 1999.

[KT] Kobrinsky, N., Trakhtenbrot, B.A.: Introduction to the Theory of Finite Automata (Russian edition, 1962), Transl. North Holland, 1965.

[M] Maler, O.: A unified approach for studying discrete and continuous dynamical systems, Proc. 37th IEEE Conf.Dec. Contr., Tampa, USA, Dec. 1998, pp. 37–42.

[MP1] Maler, O., Pnueli, A.: Timing analysis of asynchronous circuits using timed automata. Proceedings CHARME'95, LNCS 987, pp. 189–205.

[MPS] Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems, Proceedings of STACS'95, LNCS 900, pp. 229–242.

[MP2] Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems, Springer-Verlag, 1992.

[MP3] Manna, Z., Pnueli, A.: Models for reactivity. Acta Informatica 30 (1993) 609–678.

[McN] McNaughton R.: Badly timed elements and well timed nets. Technical Report 65-02, Moore-School, 1964.

[MS] Müller O., Scholz P.: Functional specification of real-time and hybrid systems, LNCS 1201, pp. 273–285, 1997.

[OD] Olderog, E.R., Dierks, H.: Decomposing real-time specifications. Intern. Symp. COMPOS'97. LNCS 1536, pp. 465–489.

[P] Pardo, D.: Timed Automata: Transducers and Circuits, M.Sc. Thesis, Tel-Aviv Univ., 1997.

[PRT] Pardo, D., Rabinovich, A., Trakhtenbrot, B.A.: On synchronous circuits over continuous time, Technical Report, Tel Aviv University, 1997.

[R] Rabinovich, A.: Finite automata over continuous time, Technical Report, Tel Aviv University, 1996.

[RT] Rabinovich, A., Trakhtenbrot, B.: From finite automata toward hybrid systems. Proceedings FCT'97, LNCS.

[RW] Ramadge P.J., Wonham W.M.: The control of discrete event systems, Proc. of the IEEE'77, pp. 81–98

[S1] Sontag, E.: Mathematical Control Theory: Deterministic Finite Dimensional Systems, Springer, N.Y., 1990.

[S2] Sontag, E.: Interconnected automata and linear systems. LNCS, No. 1066 (1996) pp. 436–448.

[T1] Trakhtenbrot, B.A.: Automata and hybrid systems. Lecture Notes on a course at Uppsala University, Fall 1997.

[T2] Trakhtenbrot, B.A.: Automata, Circuits and Hybrids: Facets of Continuous Time. Work in progress. Manuscript, pp 1-77, Draft March 2001.

[T3] Trakhtenbrot, B.A.: Origins and metamorphose of the trinity: Logics, nets, automata, Proceedings, LICS'95.

[T4] Trakhtenbrot, B.A.: Automata and their interaction: definitional suggestions, FCT99, LNCS 1684, pp. 54–89, 1999.

Languages, Rewriting Systems, and Verification of Infinite-State Systems

Ahmed Bouajjani

LIAFA laboratory, University of Paris 7,
Case 7014, 2 place Jussieu, F-75251 Paris Cedex 05, France.
abou@liafa.jussieu.fr

1 Introduction

Verification of complex systems cannot be achieved without combining several analysis methods and techniques. A widely adopted approach consists in combining abstraction methods with algorithmic verification techniques. Typically, finite abstractions are built using automatic or semi-automatic methods and model-checking algorithms are applied on these abstractions in order to verify the desired properties. However, finding faithful finite abstractions is often hard since many aspects in the behavior of the system must be hidden or encoded in a nontrivial and ad-hoc manner. This is particularly true for software systems since their behavior depends in a very crucial manner on the manipulation of data structures and variables which are assumed to range over infinite domains (e.g., unbounded stacks, queues, arrays, counters), or over finite domains whose sizes are left as parameters. Moreover, many systems are defined as networks of parametric size, i.e., they are assumed to work for an arbitrary number of processes running in parallel.

Hence, there is a real need (1) to define models allowing to capture essential aspects which are beyond the expressive power of finite models (e.g., manipulation of unbounded variables, parametrization), and (2) to develop algorithmic verification techniques which can be applied to these infinite-state models.

In this paper, we consider models based on rewriting systems and we develop an algorithmic approach for analyzing automatically such models. In the framework we adopt, configurations of systems are seen as words or vectors of words, and actions are represented by means of rewriting rules. Different rewriting policies can be considered, e.g., prefix, cyclic, or factor rewriting. They allow to model different classes of systems, e.g., pushdown systems, communicating systems through FIFO channels, or parametrized networks of identical finite-state processes connected according to a linear topology.

Then, the main problem we address is the problem of computing a representation of the infinite set of all reachable configurations in a model. Solving this problem is indeed the kernel of most of the verification methods. In our setting, this problem relies on computing the closure of a language by a rewriting system, i.e., given a rewriting system R and a language ϕ, compute $R^*(\phi)$, where R^* is the reflexive-transitive closure of the relation induced by R. We present several closure results concerning different classes of languages and rewriting systems, and we show the applications of these results in symbolic reachability analysis

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 24–39, 2001.

of different infinite-state systems. The results we present in this paper are not new. Our aim here is to present a general approach for algorithmic verification of infinite-state systems, and to show in a simple and uniform manner several results we have established in the last few years.

The paper is organized as follows: In Section 2 we present the principle of a general algorithmic verification approach based on symbolic reachability analysis. In Section 3 we define classes of rewriting systems and show their use as models for various kinds of infinite-state systems. In Section 4 we presents results on the computability of the closure of languages by rewriting systems, and show their relevance in verification. Finally, in Section 5, we give a presentation of related work.

2 Symbolic Reachability Analysis

A system can be modeled as a pair $(\mathcal{C}, \rho)$ where $\mathcal{C}$ is the set of all possible *configurations* of the system, and $\rho \subseteq \mathcal{C} \times \mathcal{C}$ is a binary *transition relation* between configurations.

Given a relation ρ, let us denote by ρ^i the relation obtained by i compositions of ρ, i.e., ρ^0 is the identity relation, and for $i \geq 0$, $\rho^{i+1} = \rho^i \circ \rho$. Then, let ρ^* be the reflexive-transitive closure of the relation ρ, i.e., $\rho^* = \bigcup_{i \geq 0} \rho^i$.

Given a relation ρ and a configuration γ, let $\rho(\gamma) = \{\gamma' \in \mathcal{C} \ : \ (\gamma, \gamma') \in \rho\}$. Intuitively, $\rho(\gamma)$ is the set of all immediate successors of the configuration γ, and $\rho^*(\gamma)$ is the set of all reachable configurations from γ. These definitions can be generalized straightforwardly to sets of configurations.

Verification problems, especially for safety properties, can be often reduced to the reachability analysis problem, i.e., to computing the set of all reachable configurations starting from a given (possibly infinite) set of *initial* configurations $\phi \subseteq \mathcal{C}$. In our setting, this consists in computing the set $\rho^*(\phi)$. More precisely, the problem is to construct a finite representation of the set $\rho^*(\phi)$, given a finite representation of the set ϕ. Then, the central problem we address can be stated as follows:

Π : *identify classes of recursive binary relations $\mathcal{R}$ between configurations as well as classes of finite representation structures $\mathcal{S}_1$ and $\mathcal{S}_2$ corresponding to two classes of sets of configurations, such that for every effectively $\mathcal{S}_1$-representable set ϕ and every relation $\rho \in \mathcal{R}$, the set $\rho^*(\phi)$ is effectively $\mathcal{S}_2$-representable.*

In order to be relevant to system verification, this problem must be addressed for classes of representation structures enjoying some minimal closure and decidability properties (e.g., the decidability of the inclusion test). Often, it is interesting to consider a stronger version of the problem above, where we require that $\mathcal{S}_1$ and $\mathcal{S}_2$ are the same class. Of course, few classes of models for practical infinite-state systems have a decidable reachability problem. Hence, it is clear that the verification problem of infinite-state systems cannot be reduced in general to finding a solution to the problem (Π). However, solutions to the problem (Π) can be embedded in a more general (or more pragmatic)

approach in order to tackle classes of infinite-state systems with an undecidable reachability problem. The idea is the following one: if we cannot provide an algorithm for computing directly the set $\rho^*(\phi)$, we adopt a semi-algorithmic approach (i.e., termination is not guaranteed) based on an iterative exploration of the reachable configurations. In order to speed up this procedure and help its termination, we use within this exploration solutions for the problem (Π) concerning subrelations of ρ. That is, at each iteration we compute the ρ-image of the reachable configurations found so far, as well as, when possible, their images by *transitive closures* of some (compositions of) subrelations of ρ. Hence, solutions for the problem (Π), even if they concern restricted classes of relations, can be relevant for enhancing the iterative reachability analysis procedure, provided that the computed intermediate sets belong to the same class of representation structures.

Let us see this in more details. From the definition of the set $\rho^*(\phi)$, a procedure for computing it would be to construct iteratively the non-decreasing sequence of sets $\phi_i = \bigcup_{0 \leq j \leq i} \rho^j(\phi)$, for $i \geq 0$, until $\phi_{k+1} \subseteq \phi_k$ for some $k \geq 0$. In such a case, we have necessarily $\phi_k = \rho^*(\phi)$. Actually, the sequence $(\phi_i)_{i \geq 0}$ can be computed by taking

$$\begin{aligned} \phi_0 &= \phi \\ \phi_{i+1} &= \phi_i \cup \rho(\phi_i) \end{aligned}$$

Of course, in order to be able to apply this procedure, we need a class of representation structures $\mathcal{S}$ such that: (1) ϕ is $\mathcal{S}$-representable, (2) $\mathcal{S}$ is effectively closed under union and computing the ρ-image, and (3) the inclusion test is decidable in $\mathcal{S}$.

However, it is obvious that this naive procedure does not terminate in general (in all nontrivial cases where $\mathcal{C}$ is infinite). Therefore, we enhance this procedure using a *fixpoint acceleration* technique, according to the terminology used in the abstract interpretation community [CC77].

Let us first introduce some notation. Given a finite set of relations $\mathcal{R}$, we denote by $Comp(\mathcal{R})$ the smallest set of relations which contains $\mathcal{R}$ and which is closed under the operations of union ($\cup$) and composition ($\circ$) of relations.

Now, let ρ be a relation and ϕ be a set of initial configurations representable in a class of representation structure $\mathcal{S}$. A new reachability analysis procedure for computing $\rho^*(\phi)$ can be defined by considering a decomposition

$$\rho = \rho' \cup \rho_1 \cup \ldots \cup \rho_m$$

and by defining a finite set of relations $\theta_1, \ldots, \theta_n \in Comp(\{\rho_1, \ldots, \rho_m\})$ such that it is possible to compute and to represent in $\mathcal{S}$ the set $\theta_i^*(\psi)$, for each $i \in \{1, \ldots, n\}$ and for every ψ in the class $\mathcal{S}$. Typically, the decomposition of ρ we consider is extracted from its definition as a finite union of relations, each of them corresponding to a possible action (or set of actions) in the modeled system, and very often, the θ_i's can be chosen to be the ρ_i's themselves.

Then, the new iterative procedure we apply consists in computing the sequence of non-decreasing sets $(\psi_i)_{i\geq 0}$ defined as follows:

$$\psi_0 = \phi$$
$$\psi_{i+1} = \psi_i \cup \rho'(\psi_i) \cup \theta_1^*(\psi_i) \cup \ldots \cup \theta_n^*(\psi_i)$$

until $\psi_{k+1} \subseteq \psi_k$ for some $k \geq 0$. Clearly, we have for every $i \geq 0$, $\phi_i \subseteq \psi_i$ and $\psi_i \subseteq \bigcup_{i\geq 0} \rho^i(\phi) = \rho^*(\phi)$. This means that, if this procedure stops, it computes precisely an $\mathcal{S}$-representation of the set $\rho^*(\phi)$.

The procedure described above generates the set of reachable configurations according to a breadth first search strategy, using additional transitions called *meta-transitions* (as in [BW94]), each of them corresponding to iterating an arbitrary number of times the application of a transition relation θ_i, for $i \in \{1, \ldots, n\}$. Actually, different search strategies may be adopted for generating the set of reachable configurations, e.g., a depth first search strategy with priority to meta-transitions.

Of course, the method proposed above does not guarantee termination. The reachability analysis procedure terminates only if we can define a suitable finite set of meta-transitions. This is obviously related to our ability to find solutions to the problem (Π) stated at the beginning of this section.

3 Models Based on Rewriting Systems

We consider here models which correspond to systems operating on sequential data structures (such as stacks or queues). In these models, configurations are vectors of words, and transition relations between configurations are defined by means of sets of rewriting rules.

3.1 Rewriting Systems

Let Σ be a finite alphabet (set of symbols). For $n \geq 1$, an n-dim *rewriting rule* r over Σ is a pair $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ where $\boldsymbol{x}, \boldsymbol{y} \in (\Sigma^*)^n$. We denote such a rule by $r : \boldsymbol{x} \mapsto \boldsymbol{y}$. The *left hand side* (resp. *right hand side*) of r, denoted by $lhs(r)$ (resp. $rhs(r)$), is the vector $\boldsymbol{x}$ (resp. $\boldsymbol{y}$).

A n-dim *rewriting system* is a finite set of n-dim rewriting rules. We consider hereafter three notions of *rewriting relations* between vectors of words. Given an n-dim rewriting system R, the prefix (resp. cyclic, factor) rewriting relation associated with R is the relation R_p (resp. R_c, R_f) $\subseteq (\Sigma^*)^n \times (\Sigma^*)^n$ such that for every $\boldsymbol{u} = (u_1, \ldots, u_n), \boldsymbol{v} = (v_1, \ldots, v_n) \in (\Sigma^*)^n$, $(\boldsymbol{u}, \boldsymbol{v}) \in R_p$ (resp. R_c, R_f) if and only if there exists a rule $r : (x_1, \ldots, x_n) \mapsto (y_1, \ldots, y_n) \in R$ such that for every $i \in \{1, \ldots, n\}$, we have respectively,

(*Prefix rewriting*) $\exists w_i \in \Sigma^*.\ u_i = x_i w_i$ and $v_i = y_i w_i$,
(*Cyclic rewriting*) $\exists w_i \in \Sigma^*.\ u_i = x_i w_i$ and $v_i = w_i y_i$,
(*Factor rewriting*) $\exists w_i, w_i' \in \Sigma^*.\ u_i = w_i x_i w_i'$ and $v_i = w_i y_i w_i'$.

3.2 Models of Infinite-State Systems

The models we consider are defined as pairs $(\mathcal{C}, \rho)$ where the set of configurations is $\mathcal{C} = (\Sigma^*)^n$, and the transition relation ρ is one of the rewriting relations $R_\dagger$, with $\dagger \in \{p, c, f\}$, for some given rewriting system R.

Automata with unbounded sequential data structures: The very common models of pushdown systems and FIFO-channel systems can be straightforwardly represented in our framework.

Indeed, prefix rewriting models the actions of a system manipulating pushdown stacks, and cyclic rewriting corresponds to operations on FIFO queues (or communication channels). One additional dimension in the rewriting system can be used to encode the control states.

For instance, consider a system which manipulates one pushdown stack (resp. one FIFO queue). A rule $r : (a, x) \mapsto (b, y)$ where $a, b \in \Sigma$ and $x, y \in \Sigma^*$, represents the action of (1) testing whether the sequence of symbols x can be removed from the stack (resp. the queue), and if yes, (2) moving from the control state a to the control state b, and putting the sequence y into the stack (resp. the queue) after having removed x from it.

In the sequel, we call an n-dim *controlled rewriting system* any set of $(n+1)$-dim rules of the form $(a, \boldsymbol{x}) \mapsto (b, \boldsymbol{y})$ where $a, b \in \Sigma$ and $\boldsymbol{x}, \boldsymbol{y} \in (\Sigma^*)^n$.

Parametrized networks: We use factor rewriting for modelling parametrized systems with an arbitrary number of identical finite-state components (processes), connected according to a linear topology.

Let Σ be the finite set of states of each of these components. Then, in order to reason uniformly about the family of systems with arbitrary number of components, we consider that a configuration is a finite word over Σ, the i^{th} element of the word corresponding to the state of the i^{th} component, and various classes of languages (e.g., regular languages) can be used to represent sets of configurations of arbitrary lengths. Therefore, actions of such parametrized systems can be represented naturally as rewriting rules, each of them corresponding to simultaneous moves in some components of the system. The kind of rules we consider here allow to represent moves involving a finite number of processes located within a bounded distance from each other. Typically, communication (rendez-vous) between immediate neighbors can be modeled by rewriting rules of the form $ab \mapsto cd$, meaning that if two processes i and $i+1$ are in states a and b respectively, then they can move simultaneously to their respective states c and d.

Take as an example a simple version of the token-passing mutual exclusion protocol: We assume that processes are arranged linearly. A process who owns the right to enter the critical section (the token) can transmit it to its right neighbor. Each process has two possible states, either 1 if it owns the token, or 0 otherwise. We suppose that initial configurations are all those in which the leftmost process has the token. Since the number of processes is not fixed, the set of initial configurations is precisely the language 10^*. Then, the transition

relation between configurations, which models the action of passing the token from left to right, corresponds to the relation R_f, where $R = \{10 \mapsto 01\}$. It is easy to see that the set of reachable configurations is $R_f^*(10^*) = 0^*10^*$.

4 Results

We present in this section solutions of the problem (Π) when the class of representation structures correspond to (subclasses of) recognizable sets. Let us recall that an n-dim recognizable set is a finite union of sets of the form $L_1 \times \ldots \times L_n$ where each L_i is a regular set (i.e., FSM definable), for $i \in \{1, \ldots, n\}$.

Clearly, the class of recognizable sets enjoys the closure and decision properties required from symbolic representation structures. Indeed, this class is closed under all boolean operations, it is also closed under the application of regular relations (notice that the relation $R_\dagger$, with $\dagger \in \{p, c, f\}$, is obviously regular for any rewriting systems R), and its inclusion problem is decidable.

4.1 Prefix Rewriting

The following theorem has been proved several times by authors from different communities with different motivations (see e.g., [Cau92, BEM97, FWW97]).

Theorem 1. *Let R be a 1-dim controlled rewriting system. Then, for every effectively recognizable set ϕ, the set $R_p^*(\phi)$ is effectively recognizable.*

In [BEM97, FWW97], versions of this theorem are used to define verification algorithms for pushdown systems against different specification logics (temporal logics and μ-calculi). Reachability analysis and model checking techniques for pushdown systems have applications in the domain of program analysis [EK99, ES01].

Unfortunately, there is no algorithm which constructs the set $R_p^*(\phi)$ for any 2-dim rewriting system R and recognizable set ϕ. This can be shown by a straightforward reduction of the Post correspondence problem.

4.2 Cyclic Rewriting

It is well known that a finite automaton equipped with a FIFO queue is as powerful as a Turing machine. So, the R_c^* image is obviously not computable for any 1-dim controlled rewriting system. Moreover, such a model can be simulated very easily by a 1-dim cyclic rewriting system: a rule of the form $(q, x) \mapsto (q', y)$ can be simulated by the application of a rule $qx \mapsto yq'$ followed by rotation rules of the form $a \mapsto a$, for all symbols a which are not control states.

Hence, in order to solve the problem (Π) for cyclic rewriting, it is necessary to restrict the considered class of rewriting systems. A typical restriction is to consider controlled rewriting systems corresponding to control loops. A *control loop* is a set of rules

$$r_1 : (q_1, \boldsymbol{x_1}) \mapsto (q_1', \boldsymbol{y_1})$$
$$\ldots$$
$$r_m : (q_m, \boldsymbol{x_m}) \mapsto (q_m', \boldsymbol{y_m})$$

such that, (1) $\forall i, j \in \{1, \ldots, m\}$ with $i \neq j$, $q_i \neq q_j$ and $q'_i \neq q'_j$, (2) $\forall i \in \{1, \ldots, m-1\}$, $q'_i = q_{i+1}$, and (3) $q'_m = q_1$.

Boigelot et al. have shown the following result:

Theorem 2 ([BGWW97]). *Let R be a 1-dim control loop. Then, for every effectively recognizable set ϕ, the set $R_c^*(\phi)$ is effectively recognizable.*

For systems of higher dimensions (even for 2-dim systems), the R_c^* image is not recognizable, in general. Indeed, consider for instance the self-loop $R = \{(q, \epsilon, \epsilon) \mapsto (q, a, a)\}$. Then, $R_c^*(q, \epsilon, \epsilon) = \{(q, a^n, a^n) \quad : \quad n \geq 0\}$ which is a non-recognizable set.

[BGWW97] provides a characterization of the control loops R such that R_c^* preserves recognizability, as well as an algorithm which constructs for such loops a finite automaton representing the R_c^* image of any given recognizable set.

In [BH97] we show that the effect of iterating any control loop can be characterized using representation structures defining a class of non-recognizable sets enjoying all needed closure and decision properties for symbolic reachability analysis. These structures, called CQDD's, correspond to a class of constrained (products of) deterministic automata. The constraints we consider are expressed in Presburger arithmetics and concern the number of times transitions of the automata are taken in the accepting runs. For instance, the set $R_c^*(q, \epsilon, \epsilon)$ above can be defined as a product of two automata A_1 and A_2 each of them recognizing the language a^*, under the constraint imposing that the number of a-transitions taken in each of the two automata are the same (see [BH97] for more details on CQDD's). We have the following result:

Theorem 3 ([BH97]). *Let R be a n-dim control loop. Then, for every effectively CQDD representable set ϕ, the set $R_c^*(\phi)$ is effectively CQDD representable.*

A similar theorem can be shown for prefix rewriting, i.e., the class of CQDD's is closed under R_p^* for any n-dim control loop R.

As mentioned in Section 3, cyclic (controlled) rewriting systems are suitable for modeling communicating systems through FIFO channels, e.g., communication protocols. In many cases, the purpose of these protocols is to ensure a perfect data transfer through unreliable channels. Hence, it is natural in this context to consider models where channels are *lossy* in the sense that they can lose a message at any time. In our setting, the lossiness assumption can be taken into account by considering a *weak* cyclic rewriting relation, where configurations can get smaller according to the subword relation (meaning that some symbols are lost), before and after any cyclic rewriting step.

Let $\preceq \subseteq \Sigma^* \times \Sigma^*$ be the subword relation, i.e., $a_1 \ldots a_n \preceq b_1 \ldots b_m$ if there exists $i_1, \ldots, i_n \in \{1, \ldots, m\}$ such that $i_1 < \ldots < i_n$ and $\forall j \in \{1, \ldots, n\}$. $a_j = b_{i_j}$. We consider the product generalization of this relation to vectors of words.

Let R be a n-dim rewriting system over Σ. We define the weak cyclic rewriting relation $R_{wc} \in (\Sigma^*)^n \times (\Sigma^*)^n)$ as follows: for every $\boldsymbol{u}, \boldsymbol{v} \in (\Sigma^*)^n$, $(\boldsymbol{u}, \boldsymbol{v}) \in R_{wc}$ if and only if there exist $\boldsymbol{u}', \boldsymbol{v}' \in (\Sigma^*)^n$ such that $\boldsymbol{u}' \preceq \boldsymbol{u}$, $\boldsymbol{v}' \in R_c(\boldsymbol{u}')$, and $\boldsymbol{v} \preceq \boldsymbol{v}'$.

An n-dim language L is *downward closed* w.r.t. the subword relation if $\forall \boldsymbol{u}, \boldsymbol{v} \in (\Sigma^*)^n$, if $\boldsymbol{v} \in L$ and $\boldsymbol{u} \preceq \boldsymbol{v}$, then $\boldsymbol{u} \in L$. Let $L_{\preceq}$ denote the *downward closure*

of L, i.e., the smallest downward closed set which includes L. Clearly, for every rewriting system R and every set ϕ, the set $R^*_{wc}(\phi)$ is downward closed. Hence, by showing that every downward closed set w.r.t $\preceq$ is a recognizable set, the following fact can be deduced.

Theorem 4 ([AČJT96,CFI96]). *For every rewriting system R, and every set ϕ, the set $R^*_{wc}(\phi)$ is a recognizable set.*

Theorem 4 does not say that the set $R^*_{wc}(\phi)$ is constructible, even though it is recognizable. Actually, we know from the results in [May00] that:

Theorem 5. *There is no algorithm which constructs the set $R^*_{wc}(\phi)$ for any given 1-dim controlled rewriting system R and recognizable set ϕ.*

We can refine Theorem 4 by defining representation structures which capture precisely the class of downward closed sets. These representation structures correspond to a particular subclass of regular expressions called *simple regular expressions* (SRE for short). Their definition is as follows: Let us call *atomic expression* any expression of the form a where $a \in \Sigma$, of the form A^* where $A \subseteq \Sigma$. A *product* is either the empty word ϵ, of a finite sequence $e_1 \cdots e_m$ of atomic expressions. Then, an SRE is either $\emptyset$, or a finite union $p_1 + \cdots + p_n$ of products. A n-dim SRE set is a finite union of Cartesian products of n SRE sets. It is very easy to see that every SRE set is downward closed w.r.t. the subword relation. Conversely, by showing that for every recognizable set L, the set $L_\preceq$ is effectively SRE representable, we obtain the following fact:

Theorem 6 ([ABJ98]). *SRE sets are precisely the downward closed sets w.r.t. the subword relation.*

The class SRE has interesting properties which makes it suitable for efficient reachability analysis of lossy FIFO-channel systems.

Theorem 7 ([ABJ98]). *The class of SRE sets is closed under union, intersection, and application of regular relations (e.g., rewriting relations). Moreover, the inclusion problem for SREs can be solved in polynomial time.*

Notice that the class of SREs is not closed under complementation. Indeed, complements of SRE languages are upward closed languages w.r.t. the subword relation. They correspond to finite unions of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \cdots a_n \Sigma^*$.

Theorem 8 ([ABJ98]). *Let R be a n-dim control loop. Then, for every SRE set ϕ, it is possible to construct an SRE representation of the set $R^*_{wc}(\phi)$ which has a polynomial size w.r.t. the size of ϕ.*

Based on the two theorems above, we derive a symbolic reachability analysis procedure as described in Section 2. This procedure has been implemented and used to analyze in a fully automatic manner several infinite-state models of communication protocols such as the alternating bit protocol, the sliding window protocol, and the bounded retransmission protocol [AAB99,ABS01].

Now, it is very easy to construct a model for which computing the effect of control loops does not help the reachability analysis procedure to terminate. Consider for instance the system R with two rules $r_1 : a \mapsto bb$ and $r_2 : b \mapsto aa$ corresponding to two self-loops (loops on a single control state q which is omitted here). It can be seen that $R^*_{wc}(a) = a^*b^* + b^*a^*$ (notice that, due to lossiness, there cannot be any constraints on the numbers of a and b in the reachable configurations). However, it is impossible to compute this set by reasoning about a finite number of iterated compositions of the two rules of R (control loops corresponding to compositions of the two considered self-loops). To see this, let us consider the relation corresponding to any of such a loop. This relation can be written as

$$\theta = \{r_2\}^{m_k}_{wc} \circ \{r_1\}^{n_k}_{wc} \circ \cdots \circ \{r_2\}^{m_1}_{wc} \circ \{r_1\}^{n_1}_{wc}$$

where the m_i's and n_i's are positive integers. It is can be checked that, for every word $w \in \Sigma^*$, $\theta^*(w)$ is always a finite language

For instance, let $w = babab$. Then, we have

$$\{r_1\}^*_{wc}(w) = \{babb^2, bb^2, bb^4\}_{\preceq} = \{babb^2, bb^4\}_{\preceq}$$
$$\{r_2\}^*_{wc}(w) = \{ababa^2, aba^2, a^2, aba^4, a^4, a^6\}_{\preceq} = \{ababa^2, aba^4, a^6\}_{\preceq}$$

Notice that the number of possible iterations of the relations $\{r_1\}_{wc}$ and $\{r_2\}_{wc}$ is always bounded. It depends on the number of occurrences of a's (resp. b's) in the initial word w.

As another example, take $\theta = \{r_2\}_{wc} \circ \{r_1\}_{wc}$ and $w = a$. Then, we have

$$\theta(w) = \{r_2\}_{wc}(\{b^2\}_{\preceq}) = \{ba^2\}_{\preceq}$$
$$\theta^2(w) = \theta(\{ba^2, a^2, ba, b, a, \epsilon\}) = \{r_2\}_{wc}(\{ab^2\}_{\preceq}) = \{ba^2\}_{\preceq}$$

Thus, we have $\theta^*(a) = \{ba^2\}_{\preceq}$.

Since $R^*_{wc}(a)$ is an infinite set, and the iteration of each relation θ of the form specified above can only produce a finite set of words, it can be concluded that the reachability analysis procedure using only meta-transitions like θ does not terminate in this case.

An interesting question is under which conditions it is possible to compute the effect of iterating *nested control loops*. Unfortunately, we have the following negative result:

Theorem 9 ([ABB01]). *There is no algorithm which constructs the set $R^*_{wc}(\phi)$ for any given 1-dim rewriting system R and any set ϕ.*

This means, that it is even impossible to compute the effect of sets of self-loops of the form $(q, x) \mapsto (q, y)$ where x and y are two words over Σ. To prove this result, we need rules where the left hand side x is of size 2. However, the situation is different when this size is assumed to be at most one.

We consider that an n-dim rewriting rule r is *context-free* if $lhs(r) \in (\Sigma \cup \{\epsilon\})^n$. A n-dim context-free rewriting system is a set of n-dim context-free rules. For instance, the system $R = \{a \mapsto bb, b \mapsto aa\}$ considered above is a context-free system. We have the following result:

Theorem 10 ([ABB01]). *Let R be a 1-dim context-free rewriting system. Then, for every effectively SRE set ϕ, the set $R^*_{wc}(\phi)$ is effectively SRE.*

Using Theorem 5, it is very easy to show that the result above cannot be extended to 2-dim context-free rewriting systems. Therefore, the question is under which conditions it is possible to construct the effect of n-dim context-free systems. We propose hereafter one such condition.

A rewriting system R is a *ring* if, for every rule $r : (x_1, \ldots, x_n) \mapsto (y_1, \ldots, y_n)$ in R, $\exists i \in \{1, \ldots, n\}$ such that $\forall j \neq i.\ x_j = \epsilon$ and $\forall j \neq (i+1) \bmod n.\ y_j = \epsilon$.

Thus, each rule r in a ring is either of the form $(\epsilon, \ldots, \epsilon, x_i, \epsilon, \ldots, \epsilon) \mapsto (\epsilon, \ldots, \epsilon, y_{i+1}, \epsilon, \ldots, \epsilon)$ or of the form $(\epsilon, \ldots, \epsilon, x_n) \mapsto (y_1, \epsilon, \ldots, \epsilon)$. Intuitively, the each rule in these systems correspond to actions of FIFO-channel systems where a word x is received from a channel of index i, and a word y is sent to the channel of index $(i+1) \bmod n$.

Theorem 11 ([ABB01]). *Let R be a n-dim context-free ring. Then, for every effectively SRE set ϕ, the set $R^*_{wc}(\phi)$ is effectively SRE.*

4.3 Factor Rewriting

As mentioned in Section 3, factor rewriting rules can be used to represent transitions in parametrized systems (networks) with an arbitrary number of identical finite-state components. An interesting class of rewriting rules which appear in this context are the so-called *semi-commutations*: A 1-dim rewriting rule is a semi-commutation if it is of the form $ab \mapsto ba$ where $a, b \in \Sigma$. A semi-commutation rewriting system is a set of semi-commutation rules.

Semi-commutations are naturally used to model transitions corresponding to information exchange between neighbors, e.g., token passing protocols for mutual exclusion (see Section 3), leader election algorithms, etc. We present later an example where semi-commutation appear in the model of a lift controller for an arbitrary number of floors. In that example, semi-commutation rules correspond to the actions of moving up or down from one floor to its immediate successor.

It is well known that the class of recognizable sets is in general not closed under R^*_f where R is any semi-commutation system. For instance, consider the system $R = \{ab \mapsto ba\}$. Then, it is easy to see that for $\phi = (ab)^*$, the set $R^*_f(\phi)$ is not recognizable.

Therefore, the question is to find a class of representation structures defining a subclass of recognizable sets which is closed under iterative semi-commutation rewriting. As an answer to this question, we propose a subclass of regular expressions called APC (alphabetic pattern constraints). We define APCs exactly as the SREs introduced above, except that we also allow in APCs atomic expressions of the form a, where $a \in \Sigma$ (APC are not downward closed w.r.t. $\preceq$ in general). In other words, APC languages are finite unions of languages of the form $\Sigma_1^* a_1 \Sigma_2^* \cdots a_n \Sigma_{n+1}^*$ where the a_i's are symbols in Σ and the Σ_i's are subsets of Σ. (The class APC coincides with the class of languages on level 3/2 of Straubing's concatenation hierarchy [PW97].)

The motivation behind the consideration of this particular kind of languages is that they appear naturally in many specification and verification contexts.

First, APC languages can be used to express properties based on specifying some patterns appearing within configurations. Typically, negations of (some) safety properties are expressed by an APC defining the set of all bad patterns. For example, in the case of the token passing protocol mentioned in Section 3, the set of bad configurations, i.e., all those which do not satisfy the mutual exclusion property, is defined by $(0+1)^*1(0+1)^*1(0+1)^*$. Thus, since this set has empty intersection with the set of reachable configurations 0^*10^*, it can be concluded that the mutual exclusion property is satisfied. Furthermore, it turns out that the reachability sets of many infinite-state systems and parametrized systems, including communication protocols like the alternation-bit and the sliding window, and parametrized mutual exclusion protocols such as the token ring, Szymanski's, Burns', or Dijkstra's protocols, are all expressible as APCs (see [ABJ98,AAB99,ABJN99,JN00,BJNT00,Tou00]).

It can be shown that the class APC has the following properties.

Theorem 12 ([BMT01]). *The class of APCs is closed under union, intersection, and rewriting (application of single rewriting rules), but it is not closed under complementation. The inclusion problem for APCs is PSPACE-complete.*

The main closure result about APCs is the following:

Theorem 13 ([Tou00,BMT01]). *Let R be a semi-commutation rewriting system. Then, for every APC set ϕ, the set $R_f^*(\phi)$ is effectively APC.*

Actually, this result can be slightly extended to system including symbol substitutions. We call a symbol substitution rewriting system any set of rules of the form $a \mapsto b$. First, it is easy to see that APCs are effectively closed under R_f^* for any symbol substitution rewriting system. The proof of Theorem 13 can be easily adapted to rewriting systems which are sets of semi-commutations and symbol substitutions [Tou00].

Let us illustrate the use of these results on a simple example. We consider a lift controller which has the following behavior: People can arrive at any time to any floor and declare their will to move up or down. The lift is initially at the lower floor, and then it keeps moving from the lower floor to the upper one, and back. In its ascending (resp. descending) phase, it takes all the people who are waiting for moving up (resp. down) and ignores the others. They are taken into account in the next phase.

For every n (number of floors), a configuration of this system can be represented by a word of the form

$$\#x_1 \cdots x_j y x_{j+1} \cdots x_n \#$$

where $y \in \{a\uparrow, a\downarrow\}$, and $x_i \in \{\bot, b\uparrow, b\downarrow, b\updownarrow\}$, for $i \in \{1, \ldots, n\}$. The symbol corresponding to x_i represents the state of the i^{th} floor: $x_i = b\updownarrow$ if there are people waiting for moving up and other people (at the same floor) waiting for moving down, $x_i = b\uparrow$ (resp. $x_i = b\downarrow$) means that there are people waiting at this floor and all of them want to move up (resp. down), and $x_i = \bot$ means that nobody is waiting at this floor. The symbol corresponding to y gives the position

of the lift: in the configuration given above, if $y = a\uparrow$ (resp. $y = a\downarrow$) then, the lift is at floor $j + 1$ (resp. j), and it is moving up (resp. down).

The set of all initial configurations, for an arbitrary number of floors, is the set of words $\phi_0 = \#a\uparrow \perp^*\#$, which means that initially, the lift is at the lower floor and there is no requests at any floor. The dynamic of the system can be modeled by the following rewriting rules:

$$\perp \mapsto b\uparrow \tag{1}$$
$$\perp \mapsto b\downarrow \tag{2}$$
$$b\uparrow \mapsto b\updownarrow \tag{3}$$
$$b\downarrow \mapsto b\updownarrow \tag{4}$$
$$a\uparrow \perp \mapsto \perp a\uparrow \tag{5}$$
$$a\uparrow b\downarrow \mapsto b\downarrow a\uparrow \tag{6}$$
$$a\uparrow b\uparrow \mapsto \perp a\uparrow \tag{7}$$
$$a\uparrow b\updownarrow \mapsto b\downarrow a\uparrow \tag{8}$$
$$a\uparrow \# \mapsto a\downarrow \# \tag{9}$$
$$\perp a\downarrow \mapsto a\downarrow \perp \tag{10}$$
$$b\uparrow a\downarrow \mapsto a\downarrow b\uparrow \tag{11}$$
$$b\downarrow a\downarrow \mapsto a\downarrow \perp \tag{12}$$
$$b\updownarrow a\downarrow \mapsto a\downarrow b\uparrow \tag{13}$$
$$\#a\downarrow \mapsto \#a\uparrow \tag{14}$$

Rules 1, 2, 3, and 4 are symbol substitutions modeling the arrival of users. Let us call request their corresponding action. Rules 5 and 6 (resp. 10 and 11) are semi-commutations modeling the moves of the lift upward (resp. downward). They correspond to the action move-up (resp. move-down). Rules 7 and 8 (resp. 12 and 13) represent the action of taking at some floor the people who want to move up (resp. down). We call the corresponding actions take-up (resp. take-down). Finally, rules 9 and 14 represent the actions of switching from the ascending to the descending phase (action up2down), and vice-versa (action down2up).

Table 1 shows the computations of the reachable configurations of the lift controller according to a depth first search strategy with priority to meta-transitions (we omit some unimportant steps). The used meta-transitions are request* corresponding to the relation $\{1 \cup 2 \cup 3 \cup 4\}_f^*$, move-up* corresponding to $\{5 \cup 6\}_f^*$, and move-down* corresponding to $\{10 \cup 11\}_f^*$. The image by request* is easy to compute (APCs are effectively closed under iterated symbol substitution rewriting), and the images by move-up* and move-down* are computable by the algorithm underlying Theorem 13.

As shown in Table 1, the reachability analysis terminates in this case thanks to the use of meta-transitions. It is worth noting that the reachability analysis procedure also gives (for free) a finite abstraction of the analyzed infinite-state model. Indeed, Table 1 defines an abstract reachability graph of the lift controller which is shown in Figure 1.

Table 1. Reachability Analysis of the Lift Controller

ϕ_0	request*	$\#a\uparrow\ (\bot + b\uparrow + b\downarrow + b\updownarrow)^*\#$	ϕ_1
ϕ_1	move-up*	$\#(\bot + b\downarrow)^* a\uparrow\ (\bot + b\uparrow + b\downarrow + b\updownarrow)^*\#$	ϕ_2
ϕ_2	request*	$\#(\bot + b\uparrow + b\downarrow + b\updownarrow)^* a\uparrow\ (\bot + b\uparrow + b\downarrow + b\updownarrow)^*\#$	ϕ_3
ϕ_3	take-up	$\#(\bot + b\uparrow + b\downarrow + b\updownarrow)^*(\bot + b\downarrow)a\uparrow\ (\bot + b\uparrow + b\downarrow + b\updownarrow)^*\#$	$\subseteq \phi_3$
ϕ_3	up2down	$\#(\bot + b\uparrow + b\downarrow + b\updownarrow)^* a\downarrow\ \#$	ϕ_4
ϕ_4	move-down*	$\#(\bot + b\uparrow + b\downarrow + b\updownarrow)^* a\downarrow\ (\bot + b\uparrow)^*\#$	ϕ_5
ϕ_5	request*	$\#(\bot + b\uparrow + b\downarrow + b\updownarrow)^* a\downarrow\ (\bot + b\uparrow + b\downarrow + b\updownarrow)^*\#$	ϕ_6
ϕ_6	take-down	$\#(\bot + b\uparrow + b\downarrow + b\updownarrow)^* a\downarrow\ (\bot + b\uparrow)(\bot + b\uparrow + b\downarrow + b\updownarrow)^*\#$	$\subseteq \phi_6$
ϕ_6	down2up	$\#a\uparrow\ (\bot + b\uparrow + b\downarrow + b\updownarrow)^*\#$	$= \phi_1$

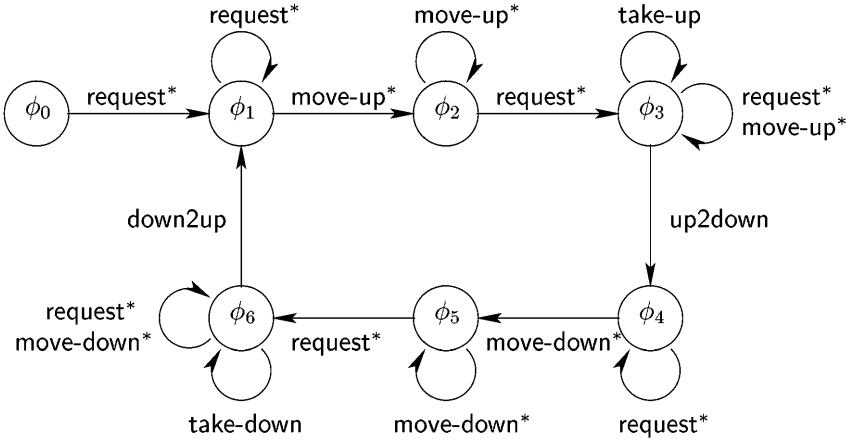


Fig. 1. Abstract Reachability Graph of the Lift Controller

5 Related Work

Several papers propose symbolic reachability analysis techniques for infinite-state systems based on using representations of languages to define sets of configurations. In these works, sets of configurations are represented by means of various kinds of automata, regular expressions, and formulas of monadic first or second order logics (see e.g., [BG96,BEM97,BH97,BGWW97,KMM+97,WB98,BJNT00,PS00,FIS00]).

Papers such as [KMM+97,WB98,BJNT00,PS00] introduce a uniform verification paradigm for infinite-state systems, called *regular model-checking*, based on the use of regular languages (finite automata or WS1S formulas) as symbolic representations, and of regular relations (finite transducers or formulas) as models of transition relations of systems. The concepts we present is this paper are very close to those developed for regular model-checking. However, we can make the following comparison between the two frameworks.

First, we do not require here that the manipulated languages are regular. For instance, the results of [BH97] show that representation structures defining non-regular languages can be used and they are needed for some applications.

Moreover, the verification approach adopted in, e.g., [BJNT00,PS00] consists in constructing (when possible) transitive closures of regular relations, (i.e., given a regular relation ρ, construct a representation of ρ^*, as a finite transducer for instance). This problem is more general and of course harder than the problem (Π) we have considered in this paper (see Section 2), which is to construct the image of a given set ϕ by ρ^*. Indeed, there are many cases where $\rho^*(\phi)$ is computable for every ϕ in some class of languages, whereas ρ^* is not constructible, or at least, not regular (e.g., for relation induced by semi-commutation rewriting systems [BMT01]). Nevertheless, in the context of regular model checking, interesting classes of relations for which the transitive closure is computable have been identified in e.g., [ABJN99,JN00]. Other works propose incomplete procedures for computing transitive closures of relations [BJNT00,PS00,DLS01].

Also, for the sake of simplicity, we have considered in this paper only special kinds of rewriting systems (for instance, these rewriting systems cannot define all the relations considered in [ABJN99,JN00,BJNT00]). Of course, more general forms of rewriting systems can be used within the framework we present.

The symbolic reachability analysis approach we describe in this paper uses the concept of meta-transition introduced in [BW94] in order to help termination. This technique can be seen as a fixpoint acceleration in the context of abstract interpretation [CC77]. However, these works use widening operators which lead in general to the computation of an upper-approximation of the reachability set, whereas the results we present in this paper allow to perform exact computations. It is worth noting that widening operations are defined depending only on the intermediary sets which are generated during the computation of the reachability set, regardless of the applied actions. In contrast, the approach we adopt here for acceleration takes into account the applied actions (rewriting rules) in order to compute the exact effect of their iteration. In [BJNT00, Tou00], widening techniques on automata and transducers are defined for regular model-checking.

The use of rewriting systems as models for infinite-state systems has been considered for instance in [Cau92,Mol96,May98]. These works address different questions from the one considered here. They are concerned with the decidability and the complexity of behavioral equivalences such as bisimulation [Cau92, Mol96] or model-checking against various propositional temporal logics [May98]. Rewriting systems are also used to model parametrized networks of identical processes in [FO97] where rewriting techniques are applied for invariant checking, but no algorithms for automatic computation of the closure of languages by rewriting systems are provided.

Finally, we have considered in this paper only rewriting systems on words. The approach we present can also be extended to rewriting systems on other structures such as trees, rings, grids, and graphs in general, in order to deal with wider classes of systems. Let us mention some of the few existing results on this topic. In [KMM+97], an extension of the regular model-checking framework to the case of tree languages is proposed in order to verify parametrized

networks with a tree-like topology. However, this paper does not provide acceleration techniques for reachability analysis. In [LS01], tree automata are used to characterize reachability sets (set of terms) for a class of processes with parallel and sequential composition which subsumes the class of context-free processes. Finally, we show in [BMT01] that Theorem 13 about closure under iterated semi-commutation rewriting can be generalized to the case of rings (circular words).

References

[AAB99] P. Abdulla, A. Annichini, and A. Bouajjani. Symbolic Verification of Lossy Channel Systems: Application to the Bounded Retransmission Protocol. In *TACAS'99*. LNCS 1579, 1999.

[ABB01] P. Abdulla, L. Boasson, and A. Bouajjani. Effective Lossy Queue Languages. In *ICALP'01*. LNCS, Springer-Verlag, 2001.

[ABJ98] P. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly Analysis of Systems with Unbounded, Lossy Fifo Channels. In *CAV'98*. LNCS 1427, 1998.

[ABJN99] P. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling Global Conditions in Parametrized System Verification. In *CAV'99*. LNCS 1633, 1999.

[ABS01] A. Annichini, A. Bouajjani, and M. Sighireanu. TReX: A Tool for Reachability Analysis of Complex Systems. In *CAV'01*. LNCS, Springer-Verlag, 2001.

[AČJT96] P. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. In *LICS'96*. IEEE, 1996.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In *CONCUR'97*. LNCS 1243, 1997.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In *CAV'96*. LNCS 1102, 1996.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *SAS'97*. LNCS 1302, 1997.

[BH97] A. Bouajjani and P. Habermehl. Symbolic Reachability Analysis of FIFO-Channel Systems with Nonregular Sets of Configurations. In *ICALP'97*. LNCS 1256, 1997. Full version in TCS **221** (1/2), pp 221-250, 1999.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular Model Checking. In *CAV'00*. LNCS 1855, 2000.

[BMT01] A. Bouajjani, A. Muscholl, and T. Touili. Permutation Rewriting and Algorithmic Verification. In *LICS'01*. IEEE, 2001.

[BW94] B. Boigelot and P. Wolper. Symbolic Verification with Periodic Sets. In *CAV'94*. LNCS 818, 1994.

[Cau92] D. Caucal. On the Regular Structure of Prefix Rewriting. *TCS*, 106(1):61–86, 1992.

[CC77] P. Cousot and R. Cousot. Static Determination of Dynamic Properties of Recursive Procedures. In *IFIP Conf. on Formal Description of Programming Concepts*. North-Holland Pub., 1977.

[CFI96] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable Channels Are Easier to Verify Than Perfect Channels. *Inform. and Comput.*, 124(1):20–31, 1996.

[DLS01] D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. In *CAV'01*. LNCS, Springer-Verlag, 2001.

[EK99] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural dataflow analysis. In *FOSSACS'99*. LNCS 1578, 1999.

[ES01] J. Esparza and S. Schwoon. A BDD-based Model Checker for Recursive Programs. In *CAV'01*. LNCS, Springer-Verlag, 2001.

[FO97] L. Fribourg and H. Olsén. Reachability sets of parametrized rings as regular languages. *Electronic Notes in Theoretical Computer Science*, 1997.

[FIS00] A. Finkel, S. Purushothaman Iyer, and G. Sutre. Well-abstracted transition systems. In *CONCUR'00*. LNCS 1877, 2000.

[FWW97] A. Finkel, B. Willems, and P. Wolper. A Direct Symbolic Approach to Model Checking Pushdown Systems. In *Infinity'97*, 1997.

[JN00] B. Jonsson and M. Nilsson. Transitive Closures of Regular Relations for Verifying Infinite-State Systems. In *TACAS'00*. LNCS 1785, 2000.

[KMM+97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *CAV'97*. LNCS 1254, 1997.

[LS01] D. Lugiez, and P. Schnoebelen. The regular viewpoint on PA-processes. In *Theoretical Computer Science*. to appear, 2001.

[May98] R. Mayr. Decidability and Complexity of Model Checking Problems for Infinite State Systems. PhD Thesis, Technische Universitaet Muenchen, April 1998.

[May00] R. Mayr. Undecidable Problems in Unreliable Computations. In *LATIN'00*. LNCS 1776, 2000.

[Mol96] F. Moller. Infinite results. In *CONCUR'96*. LNCS 1119, 1996.

[PS00] A. Pnueli and E. Shahar. Liveness and acceleration in parametrized verification. In *CAV'00*. LNCS 1855, 2000.

[PW97] J.-E. Pin and P. Weil. Polynomial closure and unambiguous product. *Theory of Computing Systems*, 30:383–422, 1997.

[Tou00] T. Touili. Vérification de Réseaux Paramétrés Basée sur des Techniques de Réécriture. MSc. Thesis (French DEA) report, Liafa Lab., University of Paris 7, July 2000. http://verif.liafa.jussieu.fr/~touili.

[WB98] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *CAV'98*. LNCS 1427, 1998.

Integrating Semantics for Object–Oriented System Models*

Martin Große–Rhode

Technische Universität Berlin, Germany, mgr@cs.tu-berlin.de,
http://tfs.cs.tu-berlin.de/~mgr/

Abstract. According to the viewpoint model of software systems development abstract models of different views of the systems are constructed. This separation of concerns reduces the complexity of the development, but prompts the question for their integration, i.e., the conception of a collection of heterogeneous models as a complete specification of a system. The integration can be achieved by using a common semantic domain for the interpretation of all models, where each viewpoint model, due to its partiality, admits a set of possible interpretations. In this paper such an integrating semantic domain is sketched and an application to structure and behaviour models of the Unified Modeling Language is discussed.

1 Introduction

The viewpoint model of software systems development that is nowadays accepted as a predominant approach to rational systems development comprises two main features. First, the development process should be based on *models*. That means, abstract representations of the functionality and behaviour of the system are provided from the very beginning and maintained during the whole life cycle of the system. This yields a concise documentation of the decisions taken in each step of the design and reduces the complexity of the development and maintenance of the system by abstraction from details. Second, these models should not represent the system in its entirety, but focus on specific aspects, like the static structure of a component or element of the system, its internal behaviour or cooperation with other components or elements of the system, etc. This distinction of *viewpoints* contributes to the separation of concerns in the development of the system and thus yields a reduction of its complexity orthogonal to the contribution of the modelling.

The viewpoint model, introduced in the Reference Model of Open Distributed Processing RM-ODP [26,23] is most prominently realized by the different diagram languages provided by the Unified Modeling Language UML [3,21], that has become the de-facto standard in object oriented modelling. It supports the

* This work has been supported by the research project IOSIP (Eh65/10-2) within the DFG Priority Programme *Integration of Software Specification Techniques for Applications in Engineering* (1064)

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 40–60, 2001.

basic distinction of structure and behaviour mentioned above, and adds further modelling languages for the implementation stage.

The separation of different viewpoints and their (independent) modelling, however, immediately prompts the question for their interrelations. On the one hand, each viewpoint model in itself only partially specifies the system, due to its focus on one aspect. This implies that the system is underspecified by each of the viewpoint models. In order to derive the complete specification of the system from the viewpoint models they have to be conceptually integrated and correspondences of the different models have to be established. On the other hand, the viewpoints are not completely orthogonal to each other. That means, certain aspects of the system will be specified by more than one model. Thus the consistency of the collection of viewpoint models has to be checked. This becomes even harder since the same aspect is specified in very different ways, using (paradigmatically) different languages. Thus even a formal definition of consistency is not obvious. Since by definition the languages will have different semantic domains one cannot request the existence of a common model as a criterion for consistency. (See for instance [2] for a discussion in the context of RM-ODP.)

In the UML the integration of the different models is addressed by the meta-modelling approach. For the definition of the languages a meta model is given that is itself a UML class diagram with constraints. The instances of the meta model are the well-formed models of the UML. Whenever model elements are instances of the same meta model element they may establish a correspondence between the models in the sense discussed above. Beyond the problem of self reference, however, i.e., defining new constructs in terms of yet undefined constructs, it is obvious that this approach addresses only the syntactic level. (The *static semantics* given by the well-formedness rules is a precondition for the semantics definition, but not the semantics definition itself.) In particular, consistency can hardly be defined or checked based on this description.

An alternative approach uses an independent semantic domain—an internal system model or a reference model—where all models can be interpreted. Obviously, such a domain must be sufficiently expressive to support the semantic interpretation of the different languages. Moreover, it must support composition operations (representing object collaboration by composition of their state machines for instance) and refinement or other development operations for the iterative and traceable development of concrete design models from abstract requirements specifications. Using a common semantic domain for the interpretation of all models the definitions of consistency and correspondence of collections of viewpoint models are supported immediately. Basically, consistency can now be reconstructed as 'having a common semantic interpretation'. However, some transformations might be required to achieve this, again due to the different viewpoints. A class diagram for instance specifies collections of objects, whereas a state machine diagram specifies single objects. Thus appropriate embeddings or projections are required that must be provided by the semantic domain. Furthermore, in a state machine events and actions may occur within

one step, whereas in a sequence diagram send actions and receipt events are different steps. Accordingly, refining or abstracting operations or relations must be provided.

An integrating semantics for all models requires to lift the interpretation of all of them to the level of full system specifications. That means, on the one hand, that an interpretation of a class diagram for instance that specifies the static structure is given by system models that also incorporate dynamic behaviour. Analogously, a state machine diagram that specifies dynamic behaviour must be interpreted with the structural aspects of data states and operation instances etc. On the other hand, the partiality of viewpoint models implies that a model will never represent a unique full system in this sense. Rather, the missing (complementary) information may be interpreted arbitrarily, which yields a set of interpretations that is admissible from the given point of view. Integration then consists of relating these sets of locally admissible interpretations of the single viewpoint models by the appropriate transformations. Their intersection then yields information about their consistency (non–empty intersection), correspondences (identical images of specification elements in common interpretations), and global under-specification (more than one model in the intersection).

In this paper a semantic approach is sketched that supports these aims. It is based on *transformation systems* that are transition systems where both states and transitions are labelled by constructs representing the corresponding structural parts. These are *data states* (attribute values, etc.) and *action structures* (event and action instances, sequences of actions, etc.) for the states and transitions respectively. Composition operations and development relations are defined for transformation systems corresponding to the requirements on an integrating semantic domain discussed above. Then the interpretation of class diagrams, state machine diagrams, and sequence diagrams and their integration is discussed. The corresponding UML definitions are used, but due to the expository character of this paper only very few concepts are considered of course. Moreover, full formal semantics are not aimed at, nor shall completely new semantics be defined in this approach. The idea is rather to use existing (formal) semantics definitions as far as possible and reconstruct them in the integrating approach presented here. The discussion on the semantics of the UML for instance is still quite open, and there are lots of efforts to define precise semantics for specific modelling languages of the UML. Quite a few approaches on the other hand address an integrating semantics for all languages. In [4,6] an approach based on stream processing functions is introduced, stressing also the set semantics of viewpoint specifications. The approach presented in [28] is based on algebras as system models. Both are to a certain extent biased in that either behaviour or structure are stressed, whereas the complementary parts have to be added or encoded into the offered constructs. One of the ideas of the transformation system approach is to reconcile these approaches.

A bridge between the meta-modelling approaches and the semantic approaches is built by the pUML group at present (see http://www.cs.york.ac.uk/puml/), in particular via its activity in the meta

modelling language development (see [8]). Precise semantics both for individual languages and the UML as a whole obtained by using other formal methods or general theoretical investigations are incorporated into the realm of the UML by meta-modelling them. In [15] for instance the formal definition of generalization defined in [7] via a transformation of UML models to Z specifications is reflected in a meta model extension. As mentioned above, further research is necessary to obtain corresponding results especially for the dynamic models of the UML.

The paper is organized as follows. In the next section transformation systems, their composition operations and development relations are introduced (see [19, 20] for more detailed presentations). The semantics for class diagrams and state machine diagrams are discussed in Sect. 3 and Sect. 4 respectively by defining the corresponding sets of admissible interpretations. For the latter also composition is discussed, i.e., the construction of systems from objects. An analogous semantic investigation of sequence diagrams is sketched in Sect. 5. Section 6 concludes.

Acknowledgments. Thanks to the members of the project IOSIP at the TU Berlin, especially Daniel Parnitzke, Jenni Tenzer, Aliki Tsiolakis, and Mesut Özhan.

2 Transformation Systems

Transformation systems are used as formal mathematical models for the representation of single dynamic entities. These can be whole systems, subsystems, objects, threads, methods, etc., which means that the granularity of the model is not prescribed by the semantic domain. Rather, the syntactic entities, i.e., the specifications according to their specific modelling techniques determine the scopes and granularities of the models.

In their formal structure transformation systems reflect the very general duality of dynamic behaviour and static structure. Basically, a transformation system is an extended labelled transition system, where both states and transitions are labelled. That means, operational models are used as first semantic level where an integration is aspired. In contrast with the denotational semantics introduced in [1] this yields a more intuitive representation with explicit modelling of the structural aspects, too. The behavioural part of a transformation system is represented by an unlabelled transition system, i.e. a *transition graph*, given by sets of *control states* and *transitions*. These are abstractions, i.e., they are just elements of a set whose construction does not matter. Control states model the possibility of state inspections and synchronization points. Transitions model the atomic, non–interruptible steps of the entity, which can also be used for the synchronization with other systems in the sense of a synchronous execution of their steps.

The transition system is labelled then by *data states* and *action structures* for control states and transitions respectively, representing their corresponding internal structures. It is important to note that these labels are not simply

given by some set, as in usual labelled transition systems. Instead, appropriate presentation means like signatures for the data states are offered to represent all structural aspects in the labels, both for states and for transitions. Thereby also languages are given to state properties of the data states. These enriched labels yield the required flexibility to focus on the behaviour or the structure of some entity, depending on the concerned modelling technique that is to be interpreted.

2.1 Data States, Transformations, and Data Spaces

The general definition of transformation systems is generic w.r.t. the concrete formal structures used to represent data states and action structures. In the most simple case a data state is conceived as a list of values for some given list of attributes (of an object) or program variables (of a procedure or a program). The *signature* that is the precondition for this definition is then given by a list of typed syntactic entities (attributes, program variables), where a fixed set of types is assumed to be given. This implicitly also yields the language for the formulation of properties of data states.

Considering partial algebras of a given algebraic signature as data states instead of lists of values allows the representation of further structural aspects. For instance, built–in or user defined static data types with their operations can be made explicit, mutable sets representing creation and deletion of elements, or parameterized attributes like arrays or abstract queries as promoted in [12] can be used. The algebraic signature also yields terms and equations or logical formulas to denote elements and state properties of these data states. Within the signature static parts like the built–in data types like integers and strings can be designated. Non–static, mutable constants yield the syntactic entities corresponding to attributes or program variables as in the list of values–data states above. The interpretation of these constants in some algebra representing a specific data state then yields the actual value of the attribute in this state.

Finally, arbitrary other structures can be used s data states, which is made precise by considering *institutions* [18] as abstract logical frameworks for data states. An institution provides signatures, and sets of sentences as well as models classes (or categories) for the signatures. The latter are related by *satisfaction relations* $\models_\Sigma$ that define whether a model M of some signature Σ satisfies a sentence φ, denoted $M \models_\Sigma \varphi$. Algebraic signatures with total or partial algebras as models and equations or conditional equations as sentences with the usual definition of satisfaction yield institutions for instance. Other examples are the above mentioned lists of values for typed syntactic entities as signatures, or other logical frameworks with signatures, sentences (formulas), models (structures), and satisfaction relations. In the context of semantics for object–oriented models *system snapshots* are used as data state models of appropriate signatures, representing sets of objects and their links (see Sect. 3).

Corresponding to the data states different action structures can be used as transition labels in a transformation system, with appropriate signatures, too. Often single actions are considered, either atomic ones given by some set or

parameterized actions like operation calls. In the latter case the signature introduces the names of the operations and their parameter type lists. Usually also an invisible action (often called ϵ or τ as in process calculi) is considered to represent internal non–observable steps of the entity. If parallel execution of actions shall be modelled within single steps sets of actions can be used, where the empty set would then correspond to the internal action. Another encapsulation is achieved by using strings of actions, representing the sequential but non–interruptible sequence of actions within one step. This is particularly important in refinements, when an abstract representation of some computation step is refined by a sequence of steps in a more concrete model. Other modelling techniques (like statecharts for instance) use further structure to decorate transitions, given by triples of events, guards, and action sequences. The duality of events and actions yields the basis for the composition of such models (cf. the discussion in Sect. 4), analogous to the duality of input and output actions in process calculi like CCS [25].

Data states and action structures together constitute the *data space* of a transformation system, representing the complete space of possible states and state changes. Therein the conjoined labels of a transition $t : c \to d$ of control states c and d yield a *data state transformation* $T : C \Rightarrow D$, given by a commencing data state C (the data state label of c), an action structure T (the label of t), and an ending data state D (the label of d). The transition $t : c \to d$ together with this transformation $T : C \Rightarrow D$ is considered as a step of the system.

2.2 Morphisms and Development Relations

Transformation systems can be related by appropriate morphisms, i.e., structure preserving mappings. Following their formal structure such a morphism is given by a graph homomorphism to relate the transition graphs and a *forgetful functor* relating the data spaces. The latter is thereby induced by a corresponding morphism of the data space signatures. These two mappings must be compatible with each other in the sense that the labels are preserved up to restriction w.r.t. the forgeful functor.

These morphism can now also be used to define development relations of software models that are formally represented as transformation systems. An injective morphism (with appropriate side conditions) $\mathbf{S} \to \mathbf{S}'$ can be interpreted as a *reduction* of $\mathbf{S}'$ by $\mathbf{S}$. The reducing system $\mathbf{S}$ may be for instance more deterministic, i.e., closer to an implementation, and have a finer internal structure (additional private variables or attributes for instance). Composing the two morphisms (transition graph homomorphism and data space forgetful functor) in opposite directions an *extension* can be modelled. The extending system offers further behaviour but preserves the behaviour of the given system, as in an inheritance relation (as discussed in [17] for instance). Finally, closure operations that yield, for instance, sequences of steps as atomic steps again can be used in combination with the extension relation to define (sequential) refinements.

2.3 Composition of Transformation Systems

The definition of the composition of transformation systems as formal models of local system components comprises two tasks. First the connection of the components has to be defined, i.e., the *architecture* of the system is specified. Second, the result of the application of the composition operation to the interconnected components has to be defined. That means, a single transformation system must be given that represents a global view on the composed system. Abstracting thus from the internal architecture of the system structural transparency is supported.

A connection relation for two transformation systems is defined by an *identification relation* for their structures and a *synchronization relation* for their behaviours. The former states which parts of the data states and actions are shared by the two systems. Shared (pervasive) static data types, shared variables, and commonly executed actions (handshakes, event/action synchronizations, etc.) are specified here. The synchronization relation states which control states are compatible with each other in the sense that they can be entered by the two components at the same time forming one global state. This contains a consistency condition of synchronization and identification relation: synchronous control states must have identical shared data parts. The synchronization of steps is represented by the synchronization relation on the transitions. Again, this must be compatible with the transformational behaviour of the synchronized steps in the sense that shared parts are transformed in the same way.

The global view of such a connection is then given as follows. The transition graph is given by all synchronous pairs of control states and transitions of the two components respectively. That means, it is a subgraph of the cartesian product of the two local transition graphs. The signature of the global data space is given by the disjoint union of the local signatures, factorized by the congruence generated by the identification relation. Correspondingly, a global data space of a control state (c_1, c_2) is given by the *amalgamation* (cf. [13]) of their local data states. That means, each constant (value), function, and carrier set (type) is taken from the local data state that provides it. If both contain the corresponding item due to the sharing expressed by the identification relation the consistency condition ensures that these coincide, i.e., the amalgamation is well defined. As global action structure for a transition (t_1, t_2) the union of the local ones is used, taking into account the identification of actions according to the identification relation.

To specify more general architectures, given by arbitrary numbers of components and connections, the categorical structure of transformation systems and morphisms is used. In fact, each connection relation as described above yields a cospan of morphisms of transformation systems. (Similar *spans*—called channels—of formal system models are used in [34,16] to describe architectures and superposition. The span-cospan duality is due to the fact that specifications are considered as opposed to models.) The global view of a composed system is then given by the pullback of the cospan, which also comprises the projections of the global view to the local components. The general mechanism for

the composition of transformation systems is accordingly given by representing the architecture by a diagram of transformation systems, specifying the components and their interconnections. The limit of such a diagram then represents the global view and its projections to the local components as above.

3 Class Diagram Semantics

After this exposition of the semantic domain we can now discuss the interpretation of software system models.

A class diagram specifies the structure of a collection of objects. That means, the attributes and the operations of the objects belonging to each of the classes are introduced as well as the structural relationships of objects of different classes. The latter are specified by inheritance relations and associations, including aggregation and composition. The behaviour of the objects is in general not constrained by a class diagram. However, object constraints can be added that have an impact on the behaviour, too, for instance as pre and post conditions of methods. Inside the UML these are formulated in the object constraints language OCL [32].

In Fig. 1 a class diagram for objects of a network protocol is shown. (The example is copied from the DHCP protocol [11] and its UML model in [30].) Servers, clients and IP addresses are introduced. A client can discover a server and request an IP address for its connection to the network from the server, provided the latter has offered it. The request is responded by an acknowledgment (ack) or not (nak), depending on the actual availability of the address.

To define the formal semantics of a class diagram—in the sense discussed in the introduction—the set of admissible interpretations as transformation systems has to be given. Since class diagrams focus on the static structure we start with the discussion of the data states and action structures and their signature.

Each single class C in the class diagram yields a data space signature ΣC as follows. The class name and each type or class used for an attribute or an operation yield a sort name. The class sorts will be interpreted as sets of references to objects of these classes in the data states of the objects. Each attribute of the class C is translated to a constant of the corresponding sort in ΣC, and for each operation op with return type t a constant $ret\_op : t$ is introduced. This is used in the data states of the active object to represent the return value of the operation in the state after its execution. Finally, for each operation op with parameter type list $t_1, \ldots, t_n$ an action name $op : t_1, \ldots, t_n$ is introduced into the action signature of ΣC.

An association as between classes C and D with role names myD and myC respectively (like the *aServer* - *aClient* association in Fig. 1) yields a further data space signature Σas with two sorts C and D and a predicate $a : C, D$. The action signature is empty. In each system state this models the links between objects of classes C and D. Furthermore constants $myD : set(D)$ and $myC : set(C)$ are added to the signatures ΣC and ΣD respectively, if navigation in the corresponding direction is possible. These represent the references to objects that

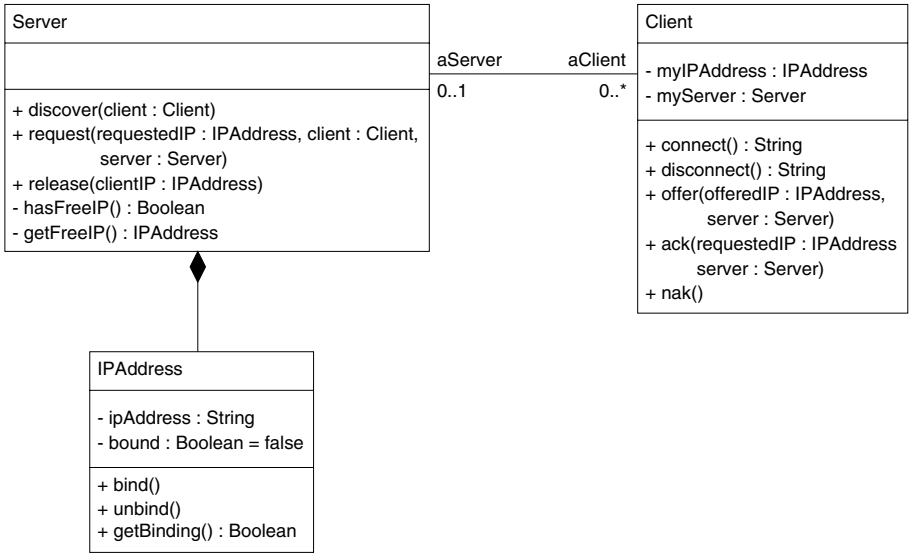


Fig. 1. Class diagram of a network protocol

are linked to the concerned object via the association *as*. Finally an inheritance relation of classes *Super* and *Sub* is mapped by adding the items of the signature $\Sigma Super$ to ΣSub and relating the two by a signature inclusion morphism.

In this way the whole class diagram is translated into a diagram of data space signatures (for details see [29]). Such a diagram is considered as the data space signature for the transformation systems that constitute the formal semantics of the class diagram. W.r.t. this signature now also the data states and transformations of an admissible transformation system are discussed.

A data state represents the state of the whole set of objects in the system at a certain point in time. Thus the following information has to be given.

- How many objects are there for each class?
- What is the state of each object, i.e., what are its attribute values and which object references does it maintain?
- How are the objects linked?

Formally a *system snapshot* is given by a tuple (I, A, L, ref) defined as follows.

- $I = (I_C)_{C \in \mathcal{C}}$ is a family of sets I_C indexed over the set $\mathcal{C}$ of classes in the class diagram. The elements of I_C are indexes to the objects of class C, that can be considered as abstract object addresses.
- $A = (A_i^C)_{i \in I_C, C \in \mathcal{C}}$ is a family of partial algebras A_i^C of signature ΣC for each object index $i \in I_C$ and class $C \in \mathcal{C}$. The algebra A_i^C represents the actual state of the object i of class C by the values of its attributes, its sets of references, etc. The association $i \rightsquigarrow A_i^C$ yields the object state associated to the index (address) i.

– For each association as a partial algebra L of signature Σas is given, whose carrier sets are given by the object index sets I_C and I_D. Thus L is basically given by a relation on these object index sets, representing the links of objects in the actual state.
– For each algebra A_i^C in A and each class sort D that occurs in ΣC a partial mapping $ref_{i,D}^C : A_{i,D}^C \dashrightarrow I_D$ is given, collected in the family ref. This represents the values of the object references the object i maintains internally in that it yields an object index (address) $ref_{i,D}^C(r) \in I_D$ of the right type for each object reference r inside i. Thereby it is required moreover that each object has a reference to itself, i.e., for each $i \in I_C$ there is a distinguished reference $self \in A_{i,C}^C$ with $ref_{i,C}^C(self) = i$.

  Note that the internal object references are elements of the algebra A_i^C, whereas the object indexes obtained by the ref–functions reside outside A_i^C.

A system snapshot conforming to the class diagram shown in Fig. 1 is shown in Fig. 2. Two *Server*–objects, two *IPAddress*–objects, and one *Client*–object

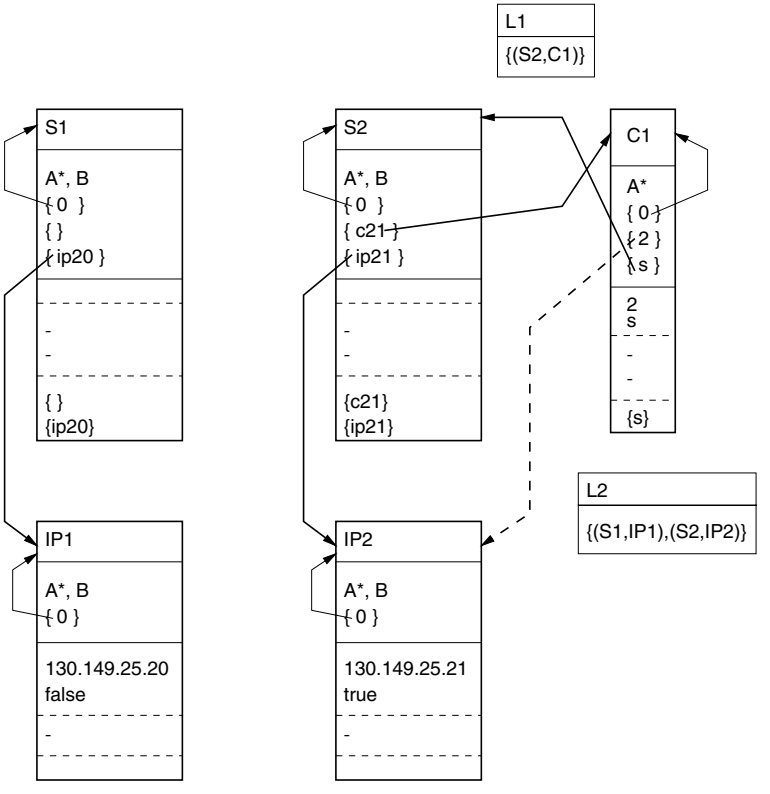


Fig. 2. Object configuration for the network protocol

are shown. The corresponding index sets are $I_{Server} = \{S1, S2\}$, $I_{IPAddress} = \{IP1, IP2\}$, and $I_{Client} = \{C1\}$ respectively. The graphical presentation of an object state as partial algebra is explained in Fig. 3 that shows the *Client*-signature and object $C1$ as its instance. The association states are depicted in Fig. 2 by the algebras (relations) $L1$ and $L2$ (the carrier sets are omitted, since they are given by the index sets). The *ref*–functions realizing the links by dereferencing are indicated by the arrows from the reference sets to the objects. Note that the link of $C1$ and $IP2$ is not supported by an association, but by the attribute *myIPAddress : IPAddress*.

		ΣClient	C1
sorts		String Client IPAddress Server	A* {0} {2} {s}
functions	attributes	myIPAddress : IPAddress myServer : Server	2 s
	return values	ret_connect :String ref_diconnect : String	- -
	associations	aServer : Set(Server)	{s}

Fig. 3. State of the Client-object $C1$ as partial ΣClient-algebra

The ingredients of a system snapshot have to obey the following consistency conditions.

- For each pair (i, i') in an association algebra L there must be a reference r in the object A_i^C such that $ref_i^C(r) = i'$ if navigation is possible in this direction, and vice versa.
- If $i \in I_{Sub}$ for some class *Sub* which is a subclass of a class *Super* then also $i \in I_{Super}$ and the reduct of A_i^{Sub} to the smaller signature $\Sigma Super$ coincides with A_i^{Super}. That means, each object of the subclass can be considered also as an object of the superclass, via the same index in another index set.
- Association multiplicities must be respected and for each (component) object of a class with a composition association to a composite class there must be a composite object it belongs to.

This defines all admissible data states of a transformation system for the class diagram.

An action structure for a transformation step from one system snapshot to another one is given by a set of operation calls. Thus parallel actions are used,

corresponding to the possible concurrency of the objects in the system. However, each operation call must be decorated with a target that allows to determine the object that executes the operation. For that purpose the object indexes are employed, using the common dot notation $i.op(p_1, \ldots, p_n)$ to indicate that object i is called to execute its operation op with the actual parameters $p_1, \ldots, p_n$. With system snapshots as data states and sets of directed operation calls as action structures a class diagram yields a data space, i.e., a space of possible system states and state transformations. Since there is no behaviour specification the set of admissible interpretations of the class diagram is given by all transformation systems with this data space. That means, arbitrary transition graphs can be chosen with labels in this data space.

4 State Machine Semantics

In this section we consider a UML modelling technique for the intra-object behaviour. State machines specify how objects of a given class react to operation calls (or other events in their environment).

Basically, a state machine is a Mealy automaton with events as input and actions as output. Its transitions may be constrained in addition by guards, i.e., boolean expressions specifying conditions on the data state of the object w.r.t. the parameters of the operation call. State machines may use several further features, like hierarchical states, parallel regions, history markers etc. Their semantics has been defined in several papers by reducing them to some kind of labelled transition system (see for instance [24,14,10]), corresponding to the different variants of state charts and state machines (see [31] for a survey). Thus as a starting point for the definition of the transformation system semantics we may assume a representation of a first level operational semantics in terms of appropriate Mealy automata (with guarded transitions) already.

The basic idea of the interpretation of state machines by transformation systems is to add the data states and action structures in all possible ways, dual to the interpretation of class diagrams. Although actions are used in state machines their effect on the data states of the objects is not specified. That means, a state machine refers to data states and data state transformations via the actions and the guards, but it does not specify them. This gives rise to a set of admissible interpretations again.

The first step in the construction of a transformation system for a state machine is to add data states D to the state machine states s, i.e., to build pairs (s, D). Such a data state $D = (A, I, ref)$ is given by

- an algebra A representing the state of the concerned object, with signature ΣC induced by the corresponding class,
- a family $I = (I_C)_{C \in \mathcal{C}}$ of object indexes as in a system snapshot; these may be used as object parameters in events and actions,
- ref–functions as above, that point to the object indexes associated by the object to its internal references.

Note that references can be dereferenced, but the object at this address cannot be accessed. That means, D represents the state of one object, including a part of the surrounding system structure, but not the whole system as in a system snapshot. Thus D can be considered as an *incomplete* system snapshot with only one object and several unbound object indexes. The *self*–reference, its corresponding index, and the association with the object state A thereby represent state machine *instances*. That means, different data states can be defined using different object index sets that then represent different instances of the same state machine, distinguished by the indexes.

The data state as data space label of a control state (s, D) within a transformation system for the state machine is given by the projection to D. (A similar construction of states as pairs of control states and data states has been used in the abstract object model introduced in [33].)

The second step of the construction of admissible transformation system interpretations of state machines consists in the definition of the transitions. For that purpose consider first a transition $e[g]/a : s_1 \rightarrow s_2$ in the state machine with event/operation call $e = op(x_1, \ldots, x_n)$ with formal parameters $x_1, \ldots, x_n$, guard/boolean expression g, and a *synchronous* operation call $a = op(p_1, \ldots, p_n)$ to another object as action, with actual or formal parameters $p_1, \ldots, p_n$. Then

- for each data state D_1 that satisfies those conditions in g that only refer to the given object (*self*),
- and each data state D_2

there may be a transition $e^+[g^-]/a^+ : (s_1, D_1) \rightarrow (s_2, D_2)$ in the transition graph of a transformation system for the state machine, where

- $e^+ = op(a_1, \ldots, a_n)$ is any instantiation of e by actual parameters,
- g^- is a corresponding instantiation of the remaining conditions of g that have not yet been evaluated, i.e., the ones that refer to other objects via navigation,
- a^+ is the instantiation of a according to the one of e^+.

The set of event instances e^+ represents the possible operation call instances as non-determinism in the transition graph. The actual calls will then be selected by other objects from this set. This selection is technically obtained by the composition of the corresponding state machines resp. the corresponding transformation system. The technique of representing input actions/events as sets of choices and communication as selection from this choice is adapted from process calculi like CCS [25] and LOTOS [5].

The effect of the execution of $op(a_1, \ldots, a_n)$ on the data state of the object is represented by the data states D_2. Since the operation semantics is not specified in the state machine any data state D_2 is admissible. The operation may be non-deterministic, represented by transitions with the same event instance e^+ to different output states. For completeness it is required that in each admissible transformation system there must be a transition for each event instance that satisfies the guard.

In addition to the transitions induced by the transitions of the state machine a transformation system for the state machine should have formal idle transitions $(s, D) \to (s, D)$ for each state (s, D). The idle transitions allow the definition of global system steps as tuples of steps of the local components. Components not taking part in a global step formally perform an idle step then (cf. Sect. 4.1).

The action structure of the transition $e^+[g^-]/a^+ : (s_1, D_1) \to (s_2, D_2)$ is given by the whole triple $e^+[g^-]/a^+$, whereby the incoming each operation callis prefixed by the index of the active object. Each event thus obtains the index of the object instantiating the state machine and executing the incoming operation calls. The complete information in the action structure is needed later for the composition with other transformation systems for state machines. The action structure of an idle transition is the internal action (empty set or τ).

For a transition $e[g]/a_1; \ldots ; a_k : s_1 \to s_2$ in the state machine with a sequence of *asynchronous* operation calls $a_1; \ldots ; a_k$ intermediate states are introduced to split the local operation execution (e) from the subsequent calls to other objects (see Fig. 4). The latter cannot change the data state of the given object, due to encapsulation, since they do not belong to the operations of the class.

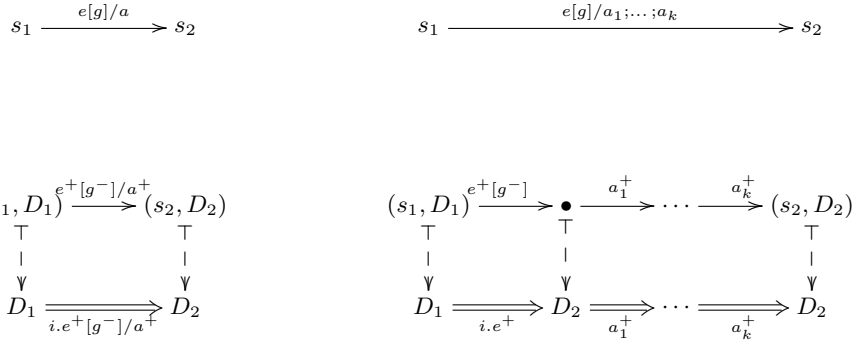


Fig. 4. Synchronous and asynchronous actions with data space attachments

Consider as example the state machines of the *Server* and *IPAddress* classes shown in Fig. 5, that are already Mealy automata. The names of the transitions represent the different behaviours of the public operations of these classes, depending on the control states of the objects and guards checking the parameters. The transition *request1* from state *hf = Has free IPAddresses* to *hf* for instance has the label

$$request(reqIP, client, server)[(server = self)\ and\ (reqIP.getBinding = false)]\ /\ reqIP.bind(); client.ack(reqIP, self)\ .$$

(The complete state machine as well as the one for the *Client*–class can be found in [30]). Transitions in a transformation system for the *Server*–state machine

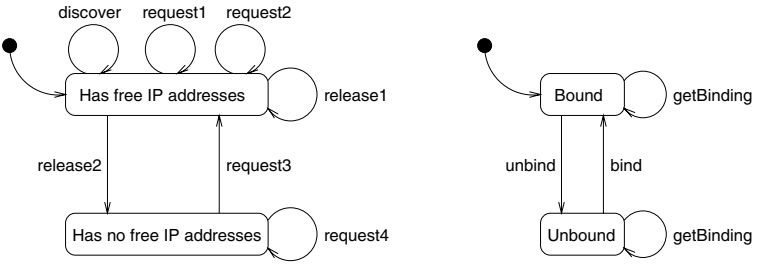


Fig. 5. State machines for Server and IPAddress objects

corresponding to the *request1* transition are shown in the central column of Fig. 6. Thereby the local data state D_{S2} is given by the state of the object $S2$ as shown in Fig. 2, with the object index sets $\{S1, S2\}, \{IP1, IP2\}, \{C1\}$ and also the *ref*-functions for $S2$ as in Fig. 2. Note that in this example the operation *request* does not change the state of the server, but only triggers further actions.

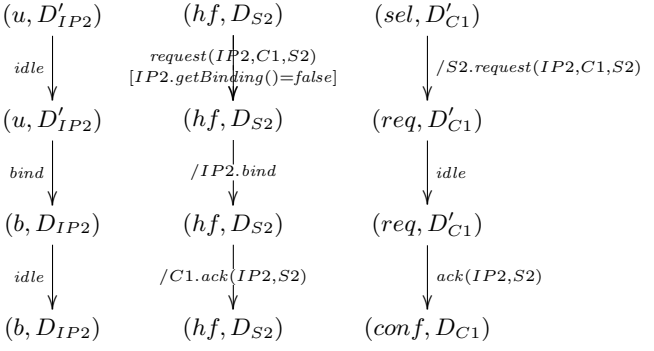


Fig. 6. Synchronous steps of IPAddress, Server, and Client objects

4.1 Composition of State Machines

The collaboration of objects whose behaviour is specified by state machines is induced by the mutual operation calls, modelled as events and actions. Semantically the corresponding composition of the state machines can be represented by the composition of transformation systems as discussed in Sect. 2.3. For that purpose an identification and a synchronization relation must be derived, which then yields the interconnection of the transformation systems as well as a global view of the common behaviour of the composition as a single object.

Consider for that purpose the steps of the *IPAddress*, *Server*, and *Client*–objects shown in Fig. 6. With the identification relation the sharing of pervasive

static data types like *Strings* and *Booleans* is expressed, i.e., they are assumed to be identical for all objects. There are no shared attributes or actions, corresponding to the object paradigm.

States (s, D) and (s', D') are synchronous if the object index sets I and I' in the data states D and D' coincide, i.e., each object refers to the same system environment. (Furthermore the interpretations of the shared static data types in the objects must coincide.) Since each object has a reference to itself the corresponding index i is already associated with the object state A_i. Together with the union of the *ref*–functions given for each object in its data state D this yields the complete set of links in the composed system.

Transitions are synchronous if their commencing and ending states are synchronous and one contains an action $a = i.op(p_1, \ldots, p_n)$ and the other one contains the complementary event $e = i.op(p_1, \ldots, p_n)$. That means, the call action is synchronized with the receipt of the call. If the operation call is synchronous this means that the induced actions of the call have to be performed in the same step and the calling object has to wait until these are finished. In an asynchronous operation call the calling object just delivers its call to finish the step. The actions take place in the subsequent steps. Beyond these synchronizations the idle transitions are synchronous with all those transitions whose commencing and ending states are synchronous with the state of the idle transition. In Fig. 6 steps of the *IPAddress*, *Server*, and *Client*–objects $IP2$, $S2$, and $C1$ are shown. In state D'_{IP2} the value of the *bound* attribute is *false*, after the execution of the *bind*–operation in state D_{IP2} it is *true*. In the initial state D_{C1} of the client the attributes *myIPAddress* and *myServer* are yet undefined. It sends a request to server $S2$ (initiated by an offer not shown in this cut), waits for an acknowledgment, and updates its attributes accordingly. The states D_{IP2}, D_{S2}, and D_{C1} are the incomplete system snapshots corresponding to the objects $IP2$, $S2$, and $C1$ as shown in Fig. 2. According to the definition given above all transitions on the same horizontal layer are thus synchronous with each other.

This composition and synchronization of state machines also allows a comparison with the class semantics. Both sets of admissible interpretations are now sets of transformation systems of the same data space (signature) except that the action structures of the composed state machine transformation system still contain the communication information (guards and actions). If a composition is considered as complete and shall be integrated with the class semantics the additional composition information has to be hidden, i.e., the labels have to be projected to the event components. The intersection of the class diagram and composed state machine semantics then yields the interpretations that are admissible from both points of view. This excludes for instance all class interpretations that do not offer the operations in the sequences specified by the state machines, and all (composed) state machine interpretations where a client has more than one server or IP–addresses are not bound to servers. These constraints are specified in the class diagram but not in the state machine diagram.

Finally, an abstraction of composed systems given by object collaborations via state machine compositions or as class diagram semantics can be given, based

on the composition operation for transformation systems. In the global view, i.e., the result of the composition operation, the composed system of Fig. 6 looks like a single object. Its class (structure) is given by the union of the three classes *Server*, *IPAddress* and *Client* (i.e., the union of their attributes, associations represented as attributes, and their operations). Each state of the global object is given by the amalgamation of the states of the local objects, including the internal object index sets as a further sort. Its actions (operation executions) are given by the union of the local actions. According to the internalization of the structure the active objects are no longer visible, i.e., the object indexes are hidden. The part of the network behaviour of the object composition $S2, IP2, C1$ for example is thus given by the sequential operation executions $request(IP2, C1, S2); bind; ack(IP2, S2)$.

5 Sequence Diagram Semantics

Sequence and collaboration diagrams in the UML specify the interaction of objects, i.e., the inter-object behaviour, via the exchange of messages. Sequence diagrams graphically stress the temporal order, whereas collaboration diagrams show the object structure corresponding to the class diagram. Semantically they are equivalent. They can be integrated into the transformation system framework easily, following the constructions and interpretations for collections of objects in the previous sections.

As starting point for the definition of the set of admissible interpretations we assume again a transition system semantics, as given for instance in [9,22] for *life sequence charts* that are a conservative extension of sequence diagrams.

For the construction of the transformation system semantics consider for example the sequence diagram in Fig. 7. This yields system snapshots and transformation steps as follows. The object instances yield object index sets $I_{Client} = \{client\}$ and $I_{Server} = \{server\}$. Arbitrary data states for the objects, i.e., $\Sigma Client$ and $\Sigma Server$–algebras, and links (*ref*-functions) supporting the message exchanges, can be added. These are not constrained by the sequence diagram. Each receipt of a message (=operation call) indicates a step of the system, given by the execution of the corresponding action. (For sake of brevity a sequential order of messages with asynchronous operation calls and message transfer is considered as example here.) Analogous to the construction of state machine transformation systems these transitions can be coupled with arbitrary data state transformations. The corresponding action labels are given by the messages of the sequence diagram, prefixed by the object instance receiving the message. We obtain thus a sequence of transformation steps $client.connect(); server.discover(client); server.getFreeIP(); \ldots$ describing a specific view of the system.

For an integration of the set of admissible interpretations of sequence diagrams obtained in this way with the class diagram semantics the object instance names have to be unified. For example, the instance names *client* and *server* used in the sequence diagram should correspond to $C1$ and $S2$ chosen in an-

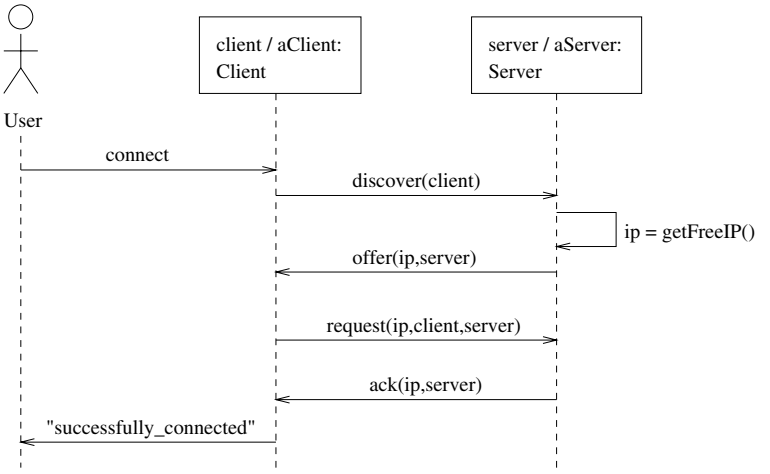


Fig. 7. A sequence diagram for the network protocol

other model. This is supported by the definition of system snapshots in the class diagram transformation systems that may comprise arbitrary object index sets. In this way correspondences between models developed independently of each other (by different persons at different sites) for example can be established.

To compare sequence diagrams with state machine diagrams and check their consistency an appropriate composition of state machine transformation systems has to be considered, corresponding to the instances defined in the sequence diagram. Vice versa, projections from the sequence diagram transformation systems to the single state machine transformation systems can be defined. The existence of a projection then proves that the sequence diagram is consistent with the state machines (see [27] for a more detailed discussion). That means, the scenario specified by the sequence diagram conforms to the capabilities of the objects as specified by their state machines.

6 Conclusion

Transformation systems have been introduced as elements of an integrating semantics for object oriented system models as given by the different diagrams of the Unified Modeling Language for instance. According to the separation of concerns realized by the viewpoint models each one only provides partial information about the system. Semantically this under-specification is reflected in defining sets of admissible interpretations for each model. These are given by transformation systems as formal mathematical representations of whole systems, incorporating their structure, behaviour, and internal interaction. The composition operations for transformation systems support the composition of components or subsystems, as for example the connection and collaboration of objects as state machine instances, as well as the abstraction from the internal

structure. That means, structural transparency is supported in the formal model much more than in most software modelling techniques. Furthermore, refinement and other development operations and relations have been defined for transformation systems. Using this integrating formal semantics these can be employed to formulate corresponding refinement relations for software system models as rules referring to the syntax of the languages directly.

The heterogeneity of the models is addressed by using one common semantic domain on the one hand, and semantic transformations corresponding to the mutual relationships of the models on the other hand. For example, state machine transformation systems that represent single object instances can be composed to obtain system models. These can then be compared with interpretations of sequence diagrams, where the number of objects is fixed. Thereby the structure that has been needed for the composition (like the distinction of events and actions and guards that may refer to the other components) must be hidden by projecting to the relevant parts. Vice versa, the system view taken by class diagrams comprises system of several objects and (degenerate) systems of single objects, which allows a comparison with the other types of diagrams.

The semantic interpretation in a common domain supports the definition of correspondences between different models that may have been developed independently of each other. At the same time, the intersection of the (appropriately transformed) sets of admissible interpretations yields a formal definition of consistency (corresponding to the correspondences established before). Note that due to the formal approach the sets are large in general, since informal meanings induced by the names used in the models cannot be taken into account. To reduce these further specification means may be considered, like object constraints in the UML. The corresponding abstract syntactic means for the representation of properties of transformation systems are introduced in [20], where also preservation results for development relations and composition operations are discussed.

The aim of these investigations conforms to the efforts of making more precise software systems modelling languages as supported by the pUML group for instance. Although the style and presentation in the orignal contributions is mathematical and thus does not conform to the UML standards, results can be achieved that are relevant and could be incorporated in appropriate meta model extensions. This could be one way to transfer theoretical results into improved software systems development standards.

References

1. M. Abadi and L. Cardelli. *A Theory of Objects*. Springer Verlag, 1996.
2. E.A. Boiten, H. Bowman, J. Derrick, P.F. Linington, and M.W.A. Steen. Viewpoint consistency in ODP. *Computer Networks*, 34(3):503–537, 2000.
3. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language, User Guide*. Addison-Wesley, 1998.
4. R. Breu, R. Grosu, F. Huber, B. Rumpe, and W. Schwerin. Systems, views and models of UML. In Martin Schader and Axel Korthaus, editors, *The Unified Modeling Language, Technical Aspects and Applications*, pages 93–109. Physica Verlag, Heidelberg, 1998.

5. Brinksma, E. (ed.). Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour. ISO 8807, 1989. International Standard.
6. M. Broy and K. Stolen. *Specification and Development of Interactive Systems. FOCUS on Streams, Interfaces, and Refinement*. Springer Verlag, 2001.
7. J.-M. Bruel and R.B. France. Transforming UML models to formal specifications. In P.-A. Muller and J. Bezivin, editors, *UML'98–Beyond the Notation*. Springer LNCS, 1998.
8. T. Clark, A. Evans, and S. Kent. The metamodelling language calculus: Foundational semantics for UML. In H. Hussmann, editor, *Proc. Fundamental Approaches to Software Engineering (FASE 2001)*, LNCS 2029, pages 17–31. Springer Verlag, 2001.
9. W. Damm and D. Harel. LSCs: Breathing Life into Message Sequence Charts. In P. Ciancarini, A. Fantechi, and R. Gorrieri, editors, *Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object–Based Distributed Systems (FMOODS'99)*, pages 293–312. Kluwer Academic Publishers, 1999.
10. W. Damm, B. Josko, H. Hungar, and A. Pnueli. A compositional real-time semantics of STATEMATE designs. In W.-P. de Roever, H. Langmaack, and A. Pnueli, editors, *Proc. COMPOS'97*, volume 1536 of *LNCS*, 1997.
11. R. Droms. Dynamic host configuration protocol. IETF, Networking Group, March 1997. Available at http://www.ietf.org/rfc/rfc2131.txt.
12. D.F. D'Souza and A.C. Wills. *Objects, Components, and Frameworks with UML. The Catalysis Approach*. Object Technology Series. Addison Wesley, 1999.
13. H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, Berlin, 1985.
14. R. Eshuis and R. Wieringa. Requirements-level semantics for UML statecharts. In S.F. Smith and C.L. Talcott, editors, *Proc. FMOODS 2000, IFIP TC6/WG6.1*, pages 121–140. Kluwer Academic Publishers, 2000.
15. A. Evans and S. Kent. Core meta-modelling semantics of UML: The pUML approach. In B. Rumpe and R.B. France, editors, *Proc. 2nd International Conference on the UML*. Springer LNCS 1723, 1999.
16. J. Fiadeiro and T. Maibaum. Temporal theories as modularisation units for concurrent system specifications. *Formal Aspects of Computing*, 4(3):239–272, 1992.
17. C. Fischer and H. Wehrheim. Behavioural subtyping relations for object-oriented formalisms. In T. Rus, editor, *Proc. Algebraic Methodology and Software Technology (AMAST 2000)*. Springer LNCS 1816, 2000.
18. J. Goguen and R.M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.
19. M. Große–Rhode. A compositional comparison of specifications of the alternating bit protocol in CCS and UNITY based on algebra transformation systems. In K. Araki, A. Galloway, and K. Taguchi, editors, *Proc. of the 1st International Conference on Integrated Formal Methods (IFM'99), York, UK, 28–29 June 1999*, pages 253–272. Springer Verlag, 1999.
20. M. Große–Rhode. Semantic integration of heterogeneous formal specifications via transformation systems. Technical report, TU Berlin, 2001.
21. Object Management Group. *Unified Modeling Language – version 1.3*, 2000. Available at http://www.omg.org/uml.
22. J. Klose and H. Wittke. An automata based interpretation of life sequence charts. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, pages 512–527. Springer LNCS 2031, 2001.

23. P.F. Linington. Introduction to the open distributed processing basic reference model. In J. de Meer, V. Heymer, and R. Roth, editors, *Proceedings IFIP TC6/WG6.4 International Workshop on Open Distributed Processing*, pages 3–13, 1991.
24. G. Lüttgen, M. von der Beeck, and R. Cleaveland. A compositional approach to statecharts semantics. Technical Report NASA/CR-2000-210086, ICASE Report No. 2000-12, Langley Research Center, 2000.
25. R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.
26. ITU-T recommendation X.901 – X.904, ISO/IEC standard 10746. Reference Model of Open Distributed Processing, 1994.
27. M. Özhan. Semantische Konsistenzanalyse von UML Sequenz- und Zustandsdiagrammen. Technical Report 2001/07, TU Berlin, 2001.
28. G. Reggio, M. Cerioli, and E. Astesiano. Towards a rigoruos semantics of UML supporting its multiview aproach. In H. Hussmann, editor, *Proc. Fundamental Approaches to Software Engineering (FASE 2001)*, LNCS 2029, pages 171–186. Springer Verlag, 2001.
29. J. Tenzer. Translation of UML class diagrams into diagrams of transformation specifications. Technical Report 2000/15, TU Berlin, FB Informatik, November 2000.
30. A. Tsiolakis. Semantic analysis and consistency checking of UML sequence diagrams. Technical Report 2001/06, TU Berlin, 2001.
31. M. von der Beeck. A comparison of Statechart variants. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Third International School and Symposium on Formal Techniques in Real-time and Fault-tolerant Systems (FTRTFT'94)*, pages 128–148. Springer LNCS 863, 1994.
32. J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1998.
33. M. Weber. Abstract object systems – a three-layer model of concurrent real-time object systems. Technical Report 97/12, TU Berlin, 1997.
34. M. Wermelinger and J.L. Fiadeiro. Algebraic software architecture reconfiguration. In *Proc. Software Engineering ESEC/FSE'99*, pages 393–409. Springer LNCS 1687, 1999.

Modelling with Partial Orders – Why and Why Not?

Extended Abstract

Mogens Nielsen

BRICS,* Dept. of Computer Science, University of Aarhus, DK

Abstract. Labelled partial orders in concurrency are a natural and powerful modelling formalism. Recently, there has been a renewed focus on such models arising in various areas of applications. We survey some results on interesting problems for partial order based models, focussing on decidability issues.

Summary

Within Concurrency a multitude of models have been suggested and studied, for a survey see [WN95]. In the report from the Concurrency Working Group from ACM Strategic Directions in Computing Research [SDCR], these models were classified on the basis of the stances they adopt with respect to three basic dichotomies:

- Intensionality versus Extensionality
- Interleaving versus True Concurrency
- Branching versus Linear Time

Most of the successful research within concurrency has been dealing with interleaving models, resulting in a great deal of insight and practical tools for reasoning about concurrent systems. However, a substantial number of results have been obtained also on so-called true concurrency or noninterleaving models. Recently, there has been a renewed focus on such models arising in quite different areas of applications. It turns out that for most of these applications, the extensional model is formally some version of labelled partial orders (pomset languages [P86]) or event structures [WN95]), whereas the intensional models come in a variety of different shapes (following the interpretation of intensionality and extensionality from the report mentioned above).

One such example is Message Sequence Charts [MSC97], a widely used formalism for describing system requirements at an early stage of development. MSC's occur in a number of different software methodologies, and have been used extensively in e.g. the development of distributed telecommunication software. Formally, an MSC is a labelled partial order. The elements of the partial

* Basic Research in Computer Science,
Centre of the Danish National Research Foundation.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 61–63, 2001.

order represent events in the form of sends and receipts of messages between different sequentially computing agents p and$q$, indicated by the labelling $p!q$ and $p?q$ respectively. The partial order represents the causal order of activity induced by the sequential ordering of events of the individual agents, combined with the ordering imposed by the sends and receipts of individual messages. The most commonly used intensional counterpart of MSC's is a so-called Message Sequence Graph, a finite graph generating a collection of MSC's. They are a powerful formalism with the consequence that many interesting questions on their behaviours (MSC's) become undecidable or untractable, see e.g. [AY99]. Recently, promising attempts have been made in order to identify a notion of regular MSC languages along with an associated set of reasoning tools [HMKT00].

Another example is recent work on partial order based models for security protocols, exemplified by the strand space model from [FHG98], the multiset rewriting model from [CDMLS00], and the Petri net model from [CW01]. As with MSC's the underlying extensional model used is a version of labelled partial orders, where the events are again representing the sends and receipts of messages amongst sequentially communicating agents. The agents either behave according to given security protocols, or play the role of intruders following certain (restricted) capabilities with respect to e.g. encryption and decryption. Compared with MSC's, the scenarios are simple, since the authentication and secrecy protocols studied are most often of bounded length. However, the labelling is relatively more complex, taking the structure of encrypted messages into account. the causal dependency amongst

In both examples above, the partial orders in modelling typically arise in an attempt to represent the causal dependency amongst events of communicating asynchronous sequential agents. The same phenomenon occurs in models for VLSI design, where it has been observed that keeping one global clock synchronised is usually the bottleneck in a processor design. Hence new architectures have been proposed and successfully applied, notably the so-called Globally Asynchronous and Locally Synchronous (GALS) architecture of [MHKEBLTP98]. A formal timed and partial order based model of the GALS principles has been studied recently in [NSS01].

So, labelled partial orders occur in many application areas as a natural modelling framework, and the reasoning about such models typically involves reasoning about the possible past and future of individual events in a computation, with the interpretation that past and future is relative to the causal structure of the computation. As indicated above, there is unfortunate evidence from theoretical research on such models that many of the natural questions to ask on partial order behaviours (of corresponding intensional models) easily become undecidable or untractable. This is the case for typical problems like model checking with respect to logics expressing partial order behaviours [AP99] and equivalence checking [JN00]. We illustrate this by a survey of results, including the identification of useful special cases which allow algorithmic reasoning.

Acknowledgements. Some of the results reported are based on joint work with Marcin Jurdcinski, Vladimiro Sassone, Jiri Srba, and Glynn Winskel.

References

[SDCR] ACM Strategic Directions in Computing Research, Concurrency Group Report, www.acm.org/surveys/sdcr, 1997.

[AP99] R. Alur, D. Peled. Undecidability of Partial Order Logics. Information Processing Letters 69, 137–143, 1999.

[AY99] R. Alur, M. Yannakakis. Model checking of message sequence charts. In Proceedings of CONCUR'99, Lecture Notes in Computer Science 1664, Springer-Verlag, 114–129, 1999.

[CDMLS00] I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In proceedings of 12-th IEEE Computer Security Foundations Workshop, 55–69, 1999.

[FHG98] F. J. T. Fabrega, J. C. Herzog, J. D. Guttman. Strand Spaces: Why is a security protocol correct? In Proceedings of 1998 IEEE Symposium on Security and Privacy, 160–171, 1998.

[CW01] F. Crazzolara, G. Winskel. Events in security protocols. BRICS Report Series RS-01-13, University of Aarhus, Denmark, 2001.

[HMKT00] J. G. Henriksen, M. Mukund, K. Narayan Kumar, and P. S. Thiagarajan. On Message Sequence Graphs and finitely generated regular MSC languages. In Proceedings of ICALP'00, Lecture Notes in Computer Science 1853, Springer-Verlag, 675–686, 2000.

[JN00] M. Jurdcinski, M. Nielsen. Hereditary history preserving bisimulation is undecidable. In Proceedings of STACS'00, Lecture Notes in Computer Science 1770, Springer-Verlag, 358–369, 2000.

[MHKEBLTP98] T. Meincke, A. Hemani, S. Kumar, P. Ellervee, J. Berg, D. Lindqvist, H. Tenhunen, A. Postula. Evaluating benefits of globally asynchronous locally synchronous VLSI architecture. In Proc. 16th Norchip, 50–57, 1998.

[MSC97] ITU–TS Recommendation Z.120: Message Sequence Chart (MSC). ITU–TS, Geneva, 1997.

[NSS01] M. Nielsen, V. Sassone, J. Srba. Towards a notion of distributed time in Petri nets. To appear in Proceedings of International Conference on Application and Theory of Petri Nets 2001, Lecture Notes in Computer Science, Springer-Verlag, 2001.

[P86] V. R. Pratt. Modelling concurrency with partial orders. International Journal of Parallel programming, 15(1), 33–71, 1986

[WN95] G. Winskel, M. Nielsen Models for Concurrency In Handbook of Logic in Computer Science, vol 4 , eds. S. Abramsky, D. M. Gabbay, T. S. E. Maibaum, Oxford University Press, 1–148, 1995.

Theoretical Aspects of Evolutionary Algorithms

Ingo Wegener*

FB Informatik, LS2, Univ. Dortmund, 44221 Dortmund, Germany
wegener@ls2.cs.uni-dortmund.de

Abstract. Randomized search heuristics like simulated annealing and evolutionary algorithms are applied successfully in many different situations. However, the theory on these algorithms is still in its infancy. Here it is discussed how and why such a theory should be developed. Afterwards, some fundamental results on evolutionary algorithms are presented in order to show how theoretical results on randomized search heuristics can be proved and how they contribute to the understanding of evolutionary algorithms.

1 Introduction

Research on the design and analysis of efficient algorithms was quite successful during the last decades. The very first successful algorithms (Dantzig's simplex algorithm for linear programming and Ford and Fulkerson's network flow algorithm) have no good performance guarantee. Later, research was focused on polynomial-time algorithms (see Cormen, Leiserson, and Rivest (1990)) and this type of research has been extended to approximation algorithms (see Hochbaum (1997)) and randomized algorithms (see Motwani and Raghavan (1995)). Indeed, designing and implementing an efficient algorithm with a proven performance guarantee is the best we can hope for when considering an algorithmic problem. This research has led to a long list of efficient problem-specific algorithms. Moreover, several paradigms of algorithms have been developed, among them divide-and-conquer, dynamic programming, and branch-and-bound. There are general techniques to design and analyze algorithms. However, these paradigms are successful only if they are realized with problem-specific modules. Besides these algorithms also paradigms for the design of heuristic algorithms have been developed like randomized local search, simulated annealing, and all types of evolutionary algorithms, among them genetic algorithms and evolution strategies. These are general classes of search heuristics with many free modules and parameters. We should distinguish problem-specific applications where we are able to choose the modules and parameters knowing properties of the considered problem and problem-independent realizations where we design a search heuristic to solve all problems of a large class of problems. We have to argue why one should investigate such a general scenario. One main point is that we obtain

* This work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center "Computational Intelligence" (531).

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 64–78, 2001.

the frame of a general search heuristic where some details may be changed in problem-specific applications. Moreover, there are at least two situations where problem-independent algorithms are of particular interest. First, in many applications, one has not enough resources (time, money, specialists,...) to design a problem-specific algorithm or problem-specific modules. Second, often we have to deal with "unknown" functions which have to be maximized. This scenario is called black box optimization. It is appropriate for technical systems with free parameters where the behavior of the system cannot be described analytically. Then we obtain knowledge about the unknown function only by "sampling". The t-th search point can be chosen according to some probability distribution which may depend on the first $t-1$ search points $x_1, \ldots, x_{t-1}$ and their function values $f(x_1), \ldots, f(x_{t-1})$. One main idea of all randomized search heuristics is to "forget" much of the known information and to make the choice of the probability distribution only dependent on the "non-forgotten" search points and their f-values.

Our focus is the maximization of pseudo-boolean functions $f : \{0,1\}^n \to \mathbb{R}$ which covers the problems from combinatorial optimization. We investigate and analyze randomized search heuristics which are designed to behave well on "many" of the "important and interesting" pseudo-boolean functions. Obviously, they cannot beat problem-specific algorithms and, also obviously, each randomized search heuristic is inefficient for most of the functions. The problem is to identify for a given randomized search heuristic classes of functions which are optimized efficiently and to identify typical functions where the heuristic fails. Such theoretical results will support the selection of an appropriate search heuristic in applications. One may also assume (or hope) that the search heuristic behaves well on a function which is "similar" to a function from a class where it is proved that the heuristic is efficient. Moreover, the proposed results lead to a better understanding of search heuristics. This again leads to the design of improved search heuristics and gives hints for a better choice of the parameters of the search heuristic. Finally, analytical results support the teaching of randomized search heuristics.

In black box optimization the black box (or oracle) answers queries x with $f(x)$ where $f : \{0,1\}^n \to \mathbb{R}$ is the function to be maximized. Since queries are expensive, the search cost is defined as the number of queries. For a fixed search heuristic let X_f be the random number of queries until "some good event" happens. The good event in this paper is that a query point is f-maximal. Then we are interested in the expected optimization time $E(X_f)$ and the success probability function $s(t) := \text{Prob}(X_f \leq t)$. This is an abstraction from the real problem, since obtaining the f-value of some optimal x does not imply that we know that x is optimal. In applications, we additionally need good stopping rules.

Our focus is on evolutionary algorithms which have been developed in the sixties of the last century and which have found many applications during the last ten years. Evolutionary algorithms are described in many monographs (Fogel (1995), Goldberg (1989), Holland (1975), Schwefel (1995)) and in a more recent

handbook (Bäck, Fogel, and Michalewicz (1997)). The experimental knowledge is immense, but the theory on evolutionary algorithms is still in its infancy. One can find several results on the one-step behavior of evolutionary algorithms, but these results most often have no implications on the expected optimization time or the success probability function. The famous schema theorem belongs to this category. There are even more results using simplifying or even unrealistic assumptions. The building-block hypothesis is such an idealized hypothesis which has turned out to be wrong in many realistic scenarios. Another idealized analysis works with "infinite populations". This makes it possible to apply methods from statistical dynamics. We claim that it is necessary to develop results on the expected optimization time and the success probability function which are not based on any assumptions, in particular, for generic variants of evolutionary algorithms and for "interesting" subclasses of functions. This does not exclude the investigation of fundamental problems without direct implications for concrete algorithms. The paper of Rabani, Rabinovich, and Sinclair (1995) is exemplary for such an approach.

In the rest of the paper, we elucidate our approach with some results. In Section 2, we introduce the simplest variant of an evolutionary algorithm, the so-called $(1+1)$EA, and in the following three sections we present results on the behavior of the $(1+1)$EA. In Section 3, we investigate monotone polynomials of bounded degree and, in Section 4, the special classes of affine functions and royal road functions. Section 5 contains an overview of further results on the $(1+1)$EA and some of its generalizations. In Section 6, we introduce a generic genetic algorithm which applies a crossover operator and discuss why it is more difficult to analyze evolutionary algorithms with crossover than evolutionary algorithms based solely on mutation and selection. Section 7 contains the first proof that crossover reduces the expected optimization time for some specific function from exponential to polynomial. We finish with some conclusions.

2 A Simple Evolutionary Algorithm

We describe the simplest variant of an evolutionary algorithm which works with population size 1 and is based solely on selection and mutation.

Algorithm 1 ($(1+1)$EA).

1.) Initialization: The current string $x \in \{0,1\}^n$ is chosen randomly using the uniform distribution.
2.) Selection for mutation: The current string x is chosen.
3.) Mutation: The offspring x' of x is created in the following way. The bits x'_i are independent and $\text{Prob}(x'_i = 1 - x_i) = p_m(n)$ (this parameter is called mutation probability).
4.) Selection of the next generation: The new current string equals x', if $f(x') \geq f(x)$, and x, otherwise.
5.) Continue at Step 2 (until some stopping criterion is fulfilled).

The generic value of $p_m(n)$ equals $1/n$ implying that, on average, one bit is flipped. Then the number of flipping bits is asymptotically Poisson distributed (with parameter 1). The algorithm can easily be generalized to larger population size μ. Then Step 2 is not trivial. The number of offsprings can be generalized to λ. There are many selection schemes for Step 4. The most prominent are $(\mu+\lambda)$-selection (the best μ of the μ parents and the λ offsprings are chosen) and (μ, λ)-selection (the best μ of the λ offsprings are chosen). These two selection schemes lead to the class of so-called evolution strategies (which have been developed for continuous search spaces $\mathbb{R}^n$). This explains the notion $(1+1)$EA for Algorithm 1 which can be interpreted as evolution strategy with population size 1. Another possibility is to interpret Algorithm 1 as a randomized hill climber, since it does not accept an offspring with a smaller f-value (fitness). A crucial point is that each $x' \in \{0,1\}^n$ has a positive probability of being created as an offspring of x. Hence, the $(1+1)$EA cannot get stuck forever in a non-optimal region. The analysis of the $(1+1)$EA is interesting, since

- the $(1+1)$EA is for many functions surprisingly efficient,
- the analysis of the $(1+1)$EA reveals many analytical tools for the analysis of more general evolutionary algorithms, and
- the $(1+1)$EA can be interpreted as evolutionary algorithm and as randomized hill climber.

The reason for larger populations is that a single search point may randomly choose "the wrong way" and may reach a region which makes it difficult to find the optimum. Working with a larger population one hopes that not all individuals of the current population go into a wrong direction and that some of them find the optimum efficiently. However, the individuals are not considered independently. If the individuals "on the wrong way" have during the following steps a larger fitness, they may drive out all individuals "on the right way" by selection. Hence, it is crucial to have a selection scheme which supports "enough" diversity in the population and which nevertheless eliminates bad individuals. Multi-start variants of the $(1+1)$EA cope in many situations with these problems, since the different runs of the $(1+1)$EA are independent. Performing $m(n)$ runs, each for $t(n)$ steps, leads to a success probability of $1-(1-s(t(n)))^{m(n)}$, if $s(t(n))$ is the success probability of a single run of the $(1+1)$EA.

3 The $(1+1)$EA on Monotone Polynomials

Pseudo-boolean functions $f : \{0,1\}^n \to \mathbb{R}$ have a unique representation as polynomials, i.e.,

$$f(x) = \sum_{A \subseteq \{1,\ldots,n\}} w_A \cdot \prod_{i \in A} x_i.$$

The degree of f is the largest size of a set A where $w_A \neq 0$. It is well known that the maximization of pseudo-boolean polynomials of degree 2 is NP-hard and Wegener and Witt (2001) have explicitly defined a degree-2 polynomial

where not only the expected optimization time of the $(1+1)$EA is exponential but also multi-start variants fail, since for some $c > 0$ the success probability after $2^{cn \log n}$ steps is $2^{-\Omega(n)}$. Such a function is almost a worst case function for the $(1+1)$EA, since the expected optimization time for each pseudo-boolean function is bounded above by $n^n = 2^{n \log n}$. This follows, since the probability to produce an optimal string within one step is always lower bounded by n^{-n}. We investigate monotone polynomials, i.e., polynomials where all weights w_A, $A \neq \emptyset$, are non-negative. The $(1+1)$EA treats zeros and ones in the same way. Therefore, our results also hold for polynomials which are obtained from monotone polynomials by replacing some variables x_i with $\overline{x_i} = 1 - x_i$. This includes all affine, i.e., degree-1 functions.

Knowing that a pseudo-boolean function is a monotone polynomial, the maximization is trivial. The all-one string always is optimal. However our motivation is black box optimization and we like to investigate the behavior of the $(1+1)$EA on monotone polynomials. This subclass of functions is interesting, since we can investigate the expected run time with respect to natural parameters, namely the input length n, the degree d, and the number N of terms with non-zero weight. Moreover, improvements are not always possible by the mutation of a small number of bits and strings with a large Hamming distance from the optimum may have much larger f-values than strings close to the optimum. It is easy to see that the degree is a crucial parameter. Garnier, Kallel, and Schoenauer (1999) have proved that the $(1+1)$EA has an expected optimization time of $\Theta(2^n)$ on the n-degree polynomial $x_1 x_2 \cdots x_n$. For this function we are searching for a needle, the all-one string 1^n, in a big haystack, namely $\{0,1\}^n$. It is obvious that such functions are difficult for black box optimization. The cited result can be extended to the general case of $N = 1$.

Lemma 1. *The expected optimization time of the $(1+1)$EA on a polynomial with $N = 1$ and degree d equals $\Theta(n2^d/d)$.*

Sketch of Proof. W.l.o.g. the polynomial equals $x_1 x_2 \cdots x_d$. The probability that at least one of the d essential bits flips in one step equals $1 - (1 - \frac{1}{n})^d = \Theta(d/n)$. Hence, the expected optimization time is by a factor of $\Theta(n/d)$ larger than the expected number of so-called active steps where one of the essential bits flips. As long as we have not found an optimal string, each new string is accepted and we have to analyze a simple Markoff chain. This can be done by standard arguments following Garnier, Kallel, and Schoenauer (1999). The expected number of active steps equals $\Theta(2^d)$ and the upper bound $O(2^d)$ holds for each initial string. □

This lemma proves that the $(1+1)$EA is efficient in the following black box scenario. We know that the function f is one of the functions which equals 1 if $x_1 = a_1, \ldots, x_d = a_d$ for some $(a_1, \ldots, a_d) \in \{0,1\}^d$ and 0 otherwise. No sampling algorithm can generate a smaller average optimization time than $(2^d + 1)/2$. We have an additional factor of $\Theta(n/d)$ for the so-called passive steps and only an additional factor of $\Theta(1)$ for active steps visiting some d-

prefix which has been visited before. Moreover, for $d = \omega(\log n)$ we cannot hope for randomized search heuristics with an expected polynomial optimization time.

The following analysis of the $(1+1)$EA on low-degree monotone polynomials shows its efficiency on a large class of interesting functions. Moreover, the proof presents typical analytical tools.

Theorem 1. *The expected optimization time of the* $(1+1)$*EA on a monotone polynomial with* N *non-vanishing terms and degree* $d \leq \log n$ *is bounded by* $O(Nn2^d/d)$*, i.e., by* $O(Nn)$ *for constant* d *and by* $O(Nn^2/\log n)$ *for all* $d \leq \log n$*.*

Sketch of Proof. Let $A(1), \ldots, A(N)$ be the N sets such that the weights $w_{A(i)}$ are non-vanishing, i.e., $w_{A(i)} > 0$, since the polynomial f is monotone. To simplify the notation we set $w_i = w_{A(i)}$ and assume w.l.o.g. that $w_1 \geq \cdots \geq w_N > 0$. A weight w_i is called active with respect to the string a, if $a_j = 1$ for all $j \in A(i)$, and w_i is called passive otherwise. The $(1+1)$EA can be described by a Markoff chain on $\{0,1\}^n$ and we have to estimate the expected time until we reach a string a such that all weights $w_1, \ldots, w_N$ are active with respect to a. The f-value of the current string is not decreasing during the search process.

A quite general technique is to partition $\{0,1\}^n$ into "fitness layers" and to estimate the expected time to leave a non-optimal layer. The choice of the layers is crucial. Here we choose $N+1$ layers $L_0, \ldots, L_N$ where

$$L_i := \{a | w_1 + \cdots + w_i \leq f(a) < w_1 + \cdots + w_{i+1}\},$$

if $i < N$, and $L_N := \{a | f(a) = w_1 + \cdots + w_N\}$ consists of all optimal strings. The search process leaves each layer at most once. If T_i is an upper bound for the expected time of leaving L_i from an arbitrary $a \in L_i$, then $T_0 + \cdots + T_{N-1}$ is an upper bound for the expected optimization time of the $(1+1)$EA on f. We prove the theorem by proving that $T_i = O(n2^d/d)$.

Let $a \in L_i$. Then, by definition, there exists some $j \leq i+1$ such that w_j is passive with respect to a. Moreover, if w_j gets active while no active w_m gets passive, we leave L_i. We assume w.l.o.g. that the monomial belonging to w_j equals $x_1 x_2 \cdots x_k$, $k \leq d$. The idea is to compare the "complicated" Markoff chain M_1 which describes the $(1+1)$EA on f starting with a and stopping when it leaves L_i with the "simple" Markoff chain M_2 which describes the $(1+1)$EA on $g(x) := x_1 x_2 \cdots x_k$ starting with a and stopping when it reaches a g-optimal string.

The analysis of M_2 (see Lemma 1) is simple, since each string is accepted until the process stops. M_1 is more complicated, since it is influenced by the other monomials. Some new strings are not accepted, since some of the active weights are deactivated. This can even happen for steps increasing the number of ones in the k-prefix of the string and in the $(n-k)$-suffix of the string. Nevertheless, since all weights are non-negative, we do not believe that this will be significant. In order to simplify the analysis we choose for each $m \in \{0, \ldots, k\}$ a string $a^m = (b^m, c^m)$ among the strings in L_i with m ones in the k-prefix b^m such that the expected time of M_1 to leave L_i when starting in a^m is maximal. Let

M_1' be the Markoff chain obtained from M_1 by replacing each string in L_i with m ones in the k-prefix with a^m. Let M_2' be the Markoff chain obtained from M_2 by replacing each string with m ones in the k-prefix with a^m. The expected stopping time of M_2' is by definition of g equal to the expected stopping time of M_2. The advantage is that M_1' and M_2' are Markoff chains on the small state space $\{0, \ldots, k\}$ representing the number of ones in the prefix.

It is sufficient to prove that for some constant $c' > 0$ the success probability of M_1' within $c'n2^d/d$ steps is bounded below by $\varepsilon > 0$, since the expected number of such phases then is bounded above by ε^{-1}. We analyze one phase of M_1' and estimate the failure probability, namely the probability of not leaving L_i.

If w_j gets active, it may happen that other weights get passive and we do not leave L_i. However, if w_j gets active, exactly all zeros in the k-prefix flip. The behavior of the bits in the $(n-k)$-suffix is independent of this event. Hence, with a probability of $(1-\frac{1}{n})^{n-k} \geq e^{-1}$ none of these bits flips implying that no active weight gets passive and we leave L_i. If one suffix bit flips in the step where w_j gets active, this is considered as a failure. If such a failure happens, the next phase can be handled in the same way, perhaps with another selected weight w_h instead of w_j. This failure event decreases the success probability of one phase at most by a factor of e^{-1}.

We want to compare M_1' with M_2'. In particular, we want to show that M_1' has a larger tendency to increase the number of ones in the k-prefix. However, this is not necessarily true if at least three bits of the k-prefix flip. Replacing 110 with 001 may increase the f-value while the inverse step decreases the f-value. Therefore, a step with at least three flipping prefix bits is considered as a failure. The failure probability for one step equals $\binom{k}{3}n^{-3} \leq d^3n^{-3}$ and the failure probability for one phase is bounded above by

$$c'n2^d d^{-1}d^3n^{-3} \leq c'd^2n^{-1} = o(1),$$

since $d \leq \log n$.

Let M_1'' and M_2'' be the Markoff chains M_1' and M_2', respectively, under the assumption that within one phase there is no step with at least three flipping prefix bits. The success probability of M_1'' and M_2'' compared with the success probability of M_1' and M_2', respectively, is decreased at most by a factor of $1-o(1)$. Let $p_1(m, m+d), d \in \{-2, -1, 0, +1, +2\}$, be the transition probabilities of M_1'' on the state space $\{0, \ldots, k\}$ and $p_2(m, m+d)$ the corresponding values of M_2''. Then

$$(1). \qquad p_1(m, m+d) \leq p_2(m, m+d), \text{ if } d \neq 0$$

The reason is that M_2'' accepts each new string. Moreover,

$$(2). \qquad p_2(m, m+d)e^{-1} \leq p_1(m, m+d), \text{ if } d > 0$$

This can be proved in the following way. Since at most two prefix bits are flipping, the number of ones in the prefix increases only if all flipping prefix bits flip from 0 to 1. If furthermore no suffix bit flips (probability at least e^{-1}), the new string

is accepted by M_1''. Finally, we have to prove a tendency of M_1'' of increasing the ones in the prefix (in comparison to M_2''). We claim that

$$\frac{p_1(m,m+d)}{p_2(m,m+d)} \geq \frac{p_1(m,m-d)}{p_2(m,m-d)}, \text{ if } 0 \leq m-d \leq m+d \leq k \tag{3}$$

Let us consider $a^m = (b^m, c^m), m \leq k$. Let $\overline{c}^m$ be any suffix. If M_1 accepts the mutated string $(b^m_-, \overline{c}^m)$ where b^m_- is obtained from b^m by flipping one (or two) ones into zeros, then M_1 accepts also $(b^m_+, \overline{c}^m)$ where b^m_+ is obtained from b^m by flipping one (or two) zeros into ones. Inequality (3) follows, since M_2 accepts $(b^m_+, \overline{c}^m)$ and $(b^m_-, \overline{c}^m)$.

The Markoff chain M_2'' can be easily analyzed using the methods of Garnier, Kallel, and Schoenauer (1999) and its generalization in the proof of Lemma 1. The Markoff chain M_1'' has the same asymptotic behavior. Inequality (1) shows that M_1'' may stay longer in some state than M_2''. However, Inequality (2) shows that the probabilities of going to a larger state are for M_1'' at most by a constant factor smaller than for M_2''. Hence, the effect of staying longer in the same state has not a big influence. Inequality (3) is the most important one. It shows that the probability of increasing the state from m to $m+d$ within one step may be decreased for M_1'' compared to M_2''. However, then the probability of decreasing the state from m to $m-d$ within one step has been decreased at least by the same factor. This implies that the expected number of active steps (changing the state) is for M_1'' smaller than for M_2''. However, the proof of this claim needs a careful analysis of the Markoff chain M_1'' which is omitted here. By Markoff's inequality, we can choose a constant c' such that the success probability of M_1'' within one phase of length $c'n2^d/d$ is at least $1/2$. This implies by our considerations that the success probability of M_1 within such a phase is at least $1/(2e) - o(1)$ which proves the theorem. □

We emphasize one main difference between the analysis of general randomized search heuristics and problem-specific algorithms. Most of the problem-specific algorithms are designed with respect to efficiency and also with respect to the aim of analyzing the algorithm. For monotone polynomials the randomized hillclimber flipping in each step exactly one random bit is not less efficient but much easier to analyze than the $(1+1)$EA. However, this hillclimber has disadvantages for other functions. It gets stuck in each local maxima while the $(1+1)$EA can escape efficiently from a local maximum if a string with at least the same f-value and short Hamming distance to the local maximum exists.

4 The (1 + 1)EA on Affine Functions and Royal Road Functions

Theorem 1 cannot be improved with respect to d, see Lemma 1 for the case $N = 1$. However, our analysis seems to be not optimal for large N. In order to leave the fitness layer L_i we have waited until one specific passive weight is turned into active. We also may leave L_i, since other weights get active. Moreover,

monomials can be correlated positively, e.g., $f(x) = 2x_1x_2 \cdots x_d + x_2x_3 \cdots x_{d+1}$. It takes some time to leave L_0, since we have to activate the first monomial. Afterwards, no step flipping one of the first d bits which all are 1 is accepted. Hence, the expected time to activate the second monomial is only $O(n)$. Because of the monotonicity of the polynomials different monomials cannot be correlated negatively. One may think that the case of independent monomials is the worst case.

Let $n = dm$. We consider monotone polynomials with m monomials with non-vanishing weights. All monomials are of degree d and depend on disjoint sets of variables. The special case of weights 1 is known as royal road function (Mitchell, Forrest, and Holland (1992)), since it has been assumed that these functions are difficult for all evolutionary algorithms without crossover and easy for genetic algorithms (which are based on crossover).

Theorem 2. *The expected optimization time of the* $(1 + 1)EA$ *on royal road functions of degree* d *is bounded by* $O(n(\log n)2^d/d)$.

Sketch of Proof. First, we consider m independent functions each consisting of one of the monomials of the royal road function with degree d. By Lemma 1 and Markoff's inequality, there is a constant c such that the success probability for a single monomial within $cn2^d/d$ steps is at least $1/2$. The success probability after $\lceil \log n \rceil + 1$ of such phases is at least $1 - 1/(2n)$ and, therefore, the probability that all monomials are optimized is at least $1/2$. This leads to the proposed upper bound in the scenario of m independent monomials. However, the $(1 + 1)$EA considers the m monomials in parallel. This causes only small differences. Steps where more active monomials are deactivated than passive monomials are activated are not accepted and monomials may be deactivated if enough passive monomials are activated. It is not difficult to prove that this increases the expected optimization time at most by a constant factor. □

A different proof method for Theorem 2 has been presented by Mitchell, Holland, and Forrest (1994). The result shows that for royal road functions there is not much room for improvements by crossover. We have seen in Section 3 that problem-independent search heuristics cannot be successful on the average with less than $(2^d + 1)/2$ steps. In particular, the improvement by any general search heuristic is bounded by a polynomial factor of $O(n(\log n)/d)$. The situation gets more difficult in the case of different weights. Then it is possible that more monomials get deactivated than activated. In a certain sense we may move far away from the optimum. This situation has been handled only in the case of affine functions, i.e., polynomials of degree 1. In this case it is not necessary to assume non-negative weights, since x_i can be replaced with $1 - x_i$.

Theorem 3. *The expected optimization time of the* $(1+1)EA$ *on affine functions is bounded by* $O(n \log n)$. *It equals* $\Theta(n \log n)$ *if all* n *degree-1 weights are non-zero.*

Idea of Proof. The full proof by Droste, Jansen, and Wegener (2001) is involved and long. W.l.o.g. $w_\emptyset = 0$ and $w_1 \geq \cdots w_n \geq 0$ where $w_i := w_{\{i\}}$. The main idea

is to measure the progress of the $(1+1)$EA with respect to the "generic" affine function

$$g(x_1, \ldots, x_n) := 2 \sum_{1 \le i \le n/2} x_i + \sum_{n/2 < i \le n} x_i.$$

This function plays the role of a potential function in the analysis of data structures and algorithms. Then successful steps ($x' \neq x$ and $f(x') \geq f(x)$ for the given affine function f) are distinguished from unsuccessful steps. The main step is to prove an upper bound of $O(1)$ on the expected number of successful steps to increase the g-value (not the f-value) of the current string. The bound on the number of unsuccessful steps follows then easily. Since the $(1+1)$EA accepts a string according to its f-value, it is possible that the g-value decreases. The idea is to design a slower Markoff chain where the g-value increases in one step by not more than 1 and where the expected gain of the g-value within one successful step is bounded below by a positive constant. Then a generalization of Wald's identity on stopping times can be proved and applied.

The lower bound is an easy application of the coupon collector's theorem. □

Up to now we were not successful to generalize this bound to monotone degree-2 polynomials. Nevertheless, we state the following conjecture.

Conjecture 1. The expected optimization time of the $(1+1)$EA on monotone polynomials of degree d is bounded by $O(n(\log n)2^d/d)$.

5 Further Results on the (1 + 1)EA and Its Generalizations

In Sections 3 and 4, we have tried to present typical methods for the analysis of the $(1+1)$EA by investigating and analyzing monotone polynomials. Wegener (2000) presents an overview on more methods. Here we mention shortly further directions of the research on the $(1+1)$EA and its generalizations. Droste, Jansen, and Wegener (1998) have investigated the behavior of the $(1+1)$EA on so-called unimodal functions, where each non-optimal string has a better Hamming neighbor. In particular, they have disproved that the $(1+1)$EA has a polynomial expected optimization time on unimodal functions. Wegener and Witt (2001) have shown for some special degree-2 polynomials and all squares of affine functions that they are easy for the multi-start variant of the $(1+1)$EA, although some of them are difficult for the $(1+1)$EA. When optimizing a single monomial $x_1 x_2 \cdots x_d$ we are exploring for a long time the plateau of strings of fitness 0 and it would be less efficient to accept only strict improvements. Jansen and Wegener (2000b) investigate the problem of exploring plateaus more generally. They also show that it is sometimes much better to accept only strict improvements. It has been conjectured that the mutation probability $1/n$ is at least almost optimal for the $(1+1)$EA and each f. This has been disproved by Jansen and Wegener (2000a) who also have shown that it can be even better to work with a dynamic $(1+1)$EA which changes its mutation probability following

a fixed schedule. This dynamic variant is analyzed for many functions by Jansen and Wegener (2001b). Further strategies to change the mutation probability are discussed by Bäck (1998).

6 A Generic Genetic Algorithm

Evolutionary algorithms based on selection and mutation only are surprisingly successful. Genetic algorithms are based on selection, mutation, and crossover and there is a community believing that crossover is the essential operator. The main variants of crossover for $(a, b) \in \{0,1\}^n \times \{0,1\}^n$ are

- one-point crossover (choose $i \in \{1, \ldots, n-1\}$ randomly and create the offspring $(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n)$) and
- uniform crossover (choose $c \in \{0,1\}^n$ randomly and create the offspring $d = (d_1, \ldots, d_n)$ where $d_i = a_i$, if $c_i = 0$ and $d_i = b_i$, if $c_i = 1$).

In order to apply crossover we need a population of size larger than 1. The main problem is to combine fitness-based selection with the preservation of enough diversity such that crossover has a chance to create strings different from those in the population. In the following it is sufficient to require that selection chooses x with at least the same probability as x' if $f(x) \geq f(x')$. This implies the same selection probabilities for x and x' if $f(x) = f(x')$. Many genetic algorithms replace a population within one step with a possibly totally different new population. It is easier to analyze so-called steady-state genetic algorithms where in each step only one offspring is created and perhaps exchanged with one member from the current population.

Algorithm 2 (Steady-state GA).

1) Initialization: The $s(n)$ members of the current population are chosen randomly and independently.
2) Branching: With probability $p_c(n)$, the new offspring is created with crossover (Steps 3.1, 3.2, 3.3) and with the remaining probability, the new offspring is created without crossover (Steps 4.1, 4.2).

3.1) Selection for crossover and mutation: A pair of strings (x, y) from the current population is chosen.
3.2) Crossover: z' is the result of crossover on (x, y).
3.3) Mutation: z is the result of mutation of z'. Go to Step 5.

4.1) Selection for mutation: A string x from the current population is chosen.
4.2) Mutation: z is the result of mutation of x.

5) Selection of the next generation: Add z to the current population and let W be the multi-set of strings in the enlarged population which have the minimal f-value and let W' be the set of strings in W which have the largest number of copies in W. Eliminate randomly one string from W' from the population to obtain the new population.
6) Continue at Step 2 (until some stopping criterion is fulfilled).

The analysis of genetic algorithms is even more difficult than the analysis of evolutionary algorithms without crossover. Although the crossover operator is in the focus of research since fourty years, there was no example known where crossover decreases the expected optimization time from exponential to polynomial. Experiments (Forrest and Mitchell (1993)) show that the $(1+1)$EA is for the royal road functions even faster than genetic algorithms. Watson (2000) presents a function where crossover probably helps. This is established by experiments and by a proof under some assumptions but not by a rigorous proof.

7 Real Royal Road Functions and the Crossover Operator

Jansen and Wegener (2001a) present the first example where crossover provably decreases the expected optimization time from exponential to polynomial. Because of the history and the many discussions on the royal road functions they have called their functions real royal road functions. For $a \in \{0,1\}^n$ let $|a| = a_1 + \cdots + a_n$ and let $b(a)$ be the block size of a, i.e. the length of the longest block consisting of ones only (the largest l such that $a_i = a_{i+1} = \cdots = a_{i+l-1} = 1$ for some i). Then

$$R_{n,m}(a) := \begin{cases} 2n^2 & \text{if } a = (1,1,\ldots,1) \\ n|a| + b(a) & \text{if } |a| \leq n-m \\ 0 & \text{otherwise.} \end{cases}$$

For a proof of the following lemma see Jansen and Wegener (2001a).

Lemma 2. *Evolutionary algorithms without crossover need with a probability exponentially close to* 1 *exponentially many steps to optimize the real royal road function* $R_{n,\lceil n/3 \rceil}$ *and with a probability of* $1 - n^{-\Omega(\log n)}$ *superpolynomially many steps to optimize* $R_{n,\lceil \log n \rceil}$.

Theorem 4. *Let* $s(n) = n$, $m = \lceil n/3 \rceil$ *and* p_c *a positive constant less than 1. Then the expected optimization time of the steady-state GA with one-point crossover on* $R_{n,m}$ *is bounded by* $O(n^4)$.

Sketch of Proof. Here we use the proof technique to describe intermediate aims and to estimate the expected time until the aim is reached. The advantage is that we can use afterwards the assumption that the last aim has been reached. Aim 1: All strings of the population have exactly $n-m$ ones or we have found the optimum.

This aim is reached in an expected number of $O(n^2)$ steps. It is very unlikely to start with strings with more than $n-m$ and less than n ones. The expected time to eliminate all these strings is $O(1)$. If we then do not find the optimum, we only have an expected waiting time of $O(n/m) = O(1)$ to increase the number of ones in the population. This is due to steps with mutation only. If the selected string has less than $n-m$ ones, there is a good chance to increase the number

of ones by a 1-bit mutation. If the selected string has exactly $n-m$ ones, there is a good chance to produce a replica.
Aim 2: All strings of the population have exactly $n-m$ ones and a block size of $n-m$ or we have found the optimum.

This aim is reached in an expected number of $O(n^3 \log n)$ steps. If we do not find the optimum, we only have to increase the sum of the block lengths of the strings of the current population. If not all strings have the same block length, it is sufficient to produce a replica of a string with a non-minimal block length. Otherwise, certain 2-bit mutations increase the block length.
Aim 3: All strings of the population have exactly $n-m$ ones, a block size of $n-m$, and each of the $m+1$ different strings with this property is contained in the population or we have found the optimum.

This aim is reached in an expected number of $O(n^4)$ steps. If we do not find the optimum, there is always at least one string in the current population such that a 2-bit mutation creates a string with $n-m$ ones and block size $n-m$ which was not in the population before.
Aim 4: The optimum is found.

This aim is reached in an expected number of $O(n^2)$ steps. This is the only phase where crossover is essential. With a probability of at least $p_c(n)/n^2$ crossover is chosen as search operator and $1^{n-m}0^m$ and 0^m1^{n-m} are selected. Then, with a probability of at least $1/3$, one-point crossover creates 1^n and finally, with a probability of at least e^{-1}, mutation preserves 1^n and we have found the optimum. □

Uniform crossover is less efficient for these functions. The probability of creating 1^n from $1^{n-m}0^m$ and 0^m1^{n-m} is only 2^{-2m}. This leads to a polynomial expected optimization time only if $m = O(\log n)$. Hence, crossover reduces the expected optimization time for some functions only from superpolynomial to polynomial. Jansen and Wegener (2001a) have presented a more complicated function where uniform crossover decreases the expected optimization time from exponential to polynomial.

One may ask what happens if we replace in the definition of $R_{n,m}$ the value of $b(a)$ by 0. Then the size of the plateau of the second-best strings increases from $m+1$ to $\binom{n}{m}$ and it is much harder to generate enough diversity. Jansen and Wegener (1999) have investigated this function. With uniform crossover and the very small crossover probability $p_c(n) = 1/(n \log^3 n)$ they could prove a polynomial expected optimization time for $m = O(\log n)$. This proof is technically much more involved than the proof of Theorem 4 and its counterpart for uniform crossover. Altogether, we have only made the first steps of analyzing genetic algorithms with crossover.

Conclusions

We have argued why one should investigate and analyze different forms of randomized search heuristics, among them evolutionary algorithms. The differences in

the analysis of problem-specific algorithms and general search heuristics for black box optimization have been discussed. Then our approach has been presented by analyzing some evolutionary algorithms on subclasses of the class of monotone polynomials and by proving for the first time that crossover can decrease the expected optimization time significantly.

References

1. Bäck, T. (1998). An overview of parameter control methods by self-adaptation in evolutionary algorithms. Fundamenta Informaticae 32, 51–66.
2. Bäck, T., Fogel, D. B., and Michalewicz, Z. (Eds.) (1997). *Handbook of Evolutionary Computation.* Oxford Univ. Press, Oxford.
3. Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms.* MIT Press.
4. Droste, S., Jansen, T., and Wegener, I. (1998). On the optimization of unimodal functions with the $(1 + 1)$ evolutionary algorithm. Proc. of PPSN V (Parallel Problem Solving from Nature), LNCS 1648, 13-22.
5. Droste, S., Jansen, T., and Wegener, I. (2001). On the analysis of the $(1 + 1)$ evolutionary algorithm. To appear: Theoretical Computer Science.
6. Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press.
7. Forrest, S., and Mitchell, M. (1993). Relative building-block fitness and the building-block hypothesis. Proc. of FOGA' 1993 (2nd Workshop Foundations of Genetic Algorithms), Morgan Kaufmann.
8. Garnier, J., Kallel, L., and Schoenauer, M. (1999). Rigorous hitting times for binary mutations. Evolutionary Computation 7, 173–203.
9. Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.
10. Hochbaum, D. S. (Ed.) (1997). *Approxiamtion Algorithms for NP-Hard Problems.* PWS Publ. Co.
11. Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* The University of Michigan Press.
12. Jansen, T., and Wegener, I. (1999). On the analysis of evolutionary algorithms – a proof that crossover really can help. Proc. of ESA'99 (European Symp. on Algorithms), LNCS 1643, 184–193.
13. Jansen, T., and Wegener, I. (2000a). On the choice of the mutation probability for the $(1 + 1)$EA. Proc. of PPSN VI (Parallel Problem Solving from Nature), LNCS 1917, 89–98.
14. Jansen, T., and Wegener, I. (2000b). Evolutionary algorithms – how to cope with plateaus of constant fitness and when to reject strings of the same fitness. To appear: IEEE Trans. on Evolutionary Computation.
15. Jansen, T., and Wegener, I. (2001a). Real royal road functions – where crossover provably is essential. To appear: GECCO'2001.
16. Jansen, T., and Wegener, I. (2001b). On the analysis of a dynamic evolutionary algorithm. Submitted: ESA'2001.
17. Mitchell, M., Forrest, S., and Holland, J. H. (1992). The Royal Road function for genetic algorithms: Fitness landscapes and GA performance. Proc. of 1st European Conf. on Artificial Life, 245–254, MIT Press.

18. Mitchell, M., Holland, J. H., and Forrest, S. (1994). When will a genetic algorithm outperform hill climbing. In J. Cowan, G. Tesauro, and J. Alspector (Eds.): Advances in Neural Information Processing Systems. Morgan Kaufman.
19. Motwani, R., and Raghavan, P. (1995). *Randomized Algorithms.* Cambridge Univ. Press.
20. Rabani, Y., Rabinovich, Y., and Sinclair, A. (1998). A computational view of population genetics. Random Structures and Algorithms 12, 314–334.
21. Schwefel, H.-P. (1995). *Evolution and Optimum Seeking.* Wiley.
22. Watson, R. A. (2000). Analysis of recombinative algorithms on a non-separable building-block problem. Proc. of FOGA'2000 (6. Workshop Foundations of Genetic Algorithms), to appear.
23. Wegener, I. (2000). On the expected runtime and the success probability of evolutionary algorithms. Proc. of WG'2000 (26. Workshop on Graph-Theoretic Concepts in Computer Science), LNCS 1928, 1–10.
24. Wegener, I., and Witt, C. (2001). On the analysis of a simple evolutionary algorithm on quadratic pseudo-boolean functions. Submitted: Journal of Discrete Algorithms.

Improvements of the Alder–Strassen Bound: Algebras with Nonzero Radical

Markus Bläser

Institut für Theoretische Informatik, Med. Universität zu Lübeck
Wallstr. 40, 23560 Lübeck, Germany
blaeser@tcs.mu-luebeck.de

Abstract. Let $C(A)$ denote the multiplicative complexity of a finite dimensional associative k-algebra A.
For algebras A with nonzero radical $\operatorname{rad} A$, we exhibit several lower bound techniques for $C(A)$ that yield bounds significantly above the Alder–Strassen bound. In particular, we prove that the multiplicative complexity of the multiplication in the algebras $k[X_1, \ldots, X_n]/\, I_{d+1}(X_1, \ldots, X_n)$ is bounded from below by $3 \cdot \binom{n+d}{n} - \binom{n+\lceil d/2 \rceil}{n} - \binom{n+\lfloor d/2 \rfloor}{n}$, where $I_d(X_1, \ldots, X_n)$ denotes the ideal generated by all monomials of degree d in $X_1, \ldots, X_n$. Furthermore, we show the lower bound $C(\mathrm{T}_n(k)) \geq (2\frac{1}{8} - o(1)) \dim \mathrm{T}_n(k)$ for the multiplication of upper triangular matrices.

1 Introduction

A fundamental problem in algebraic complexity theory is the question about the costs of multiplication, say of matrices, triangular matrices, polynomials, or power series, just to mention a few. To be more specific, let A be a finite dimensional associative k-algebra with unity 1. By fixing a basis of A, say $v_1, \ldots, v_N$, we can define a set of bilinear forms corresponding to the multiplication in A. If $v_\mu v_\nu = \sum_{\kappa=1}^{N} \alpha_{\mu,\nu}^{(\kappa)} v_\kappa$ for $1 \leq \mu, \nu \leq N$ with *structural constants* $\alpha_{\mu,\nu}^{(\kappa)} \in k$, then these constants and the identity

$$\left(\sum_{\mu=1}^{N} X_\mu v_\mu \right) \left(\sum_{\nu=1}^{N} Y_\nu v_\nu \right) = \sum_{\kappa=1}^{N} b_\kappa(X, Y) v_\kappa$$

define the desired bilinear forms $b_1, \ldots, b_N$. The *multiplicative complexity* of $b_1, \ldots, b_N$ is the smallest number of *essential* multiplications and divisions necessary and sufficient to compute $b_1, \ldots, b_N$ from the indeterminates $X_1, \ldots, X_N$ and $Y_1, \ldots, Y_N$.

According to Strassen [15], we may reformulate the problem over infinite fields as follows: the multiplicative complexity of $b_1, \ldots, b_N$ is the smallest number ℓ of products $p_\lambda = u_\lambda(X_i, Y_j) \cdot v_\lambda(X_i, Y_j)$ with linear forms u_λ and v_λ in the X_i and Y_j such that the linear span of $p_1, \ldots, p_\ell$ contains $b_1, \ldots, b_N$. (The restriction to infinite fields is not critical for this work, since we are concerned

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 79–91, 2001.

with lower bounds.) From this characterization, it follows that the multiplicative complexity of $b_1, \dots, b_N$ does not depend on the choice of $v_1, \dots, v_N$, thus we may speak about the multiplicative complexity of (the multiplication in) A. For a modern introduction to algebraic complexity theory, we recommend [8].

A fundamental lower bound for the multiplicative complexity is the so-called Alder–Strassen bound (1) (see Section 1.2). Recently, this bound has been improved for a large class of semisimple[1] algebras [6] as shown in (2). The main contributions of this work are improvements of the Alder–Strassen bound for algebras with nonzero radical, like upper triangular matrices.

Closely related to the multiplicative complexity is the *bilinear complexity* (or *rank*). Here the products $p_\lambda = u_\lambda(X_i) \cdot v_\lambda(Y_j)$ are bilinear products, that is, products of linear forms u_λ in the X_i and linear forms v_λ in the Y_j. (Note that $b_1, \dots, b_N$ are bilinear forms.) The multiplicative complexity is clearly a lower bound for the bilinear complexity and it is easy to show that twice the multiplicative complexity is an upper bound for the bilinear complexity (see e.g. [8, Eq. 14.8]). Therefore, we usually want to derive upper bounds for the bilinear complexity and lower bounds for the multiplicative complexity.

While the difference between multiplicative and bilinear complexity seems to be minor at a first glance, it is much harder to cope with the multiplicative complexity when dealing with lower bounds. The main reason is the fact that the bilinear complexity of a tensor of a bilinear map (see below for a definition) is invariant under permutations whereas the multiplicative complexity might not, see also [8, Chap. 14.2] for a further discussion. For instance, if we consider bilinear complexity, then the bound (2) holds also for any algebra with nonzero radical, provided that the semisimple quotient algebra $A/\operatorname{rad} A$ fulfils the corresponding premises [4,5]. On the other hand, there are examples given in [6, Sect. 6] that show that the novel methods from [6] for the multiplicative complexity do not apply to algebras with nonzero radical.

1.1 Model of Computation

For proving lower bounds, a coordinate-free definition of multiplicative complexity is often more appropriate than the one given above, see e.g. [8, Chap. 14.1]. In the following, if V is a vector space, let V^* denote the dual space of V, i.e., the vector space of all linear forms on V.

Definition 1. *Let k be a field, U, V, and W finite dimensional vector spaces over k, and $\phi : U \times V \to W$ a bilinear map.*

1. *A sequence $\beta = (f_1, g_1, w_1, \dots, f_\ell, g_\ell, w_\ell)$ with $f_\lambda, g_\lambda \in (U \times V)^*$ and $w_\lambda \in W$ is called a quadratic computation for ϕ over k of length ℓ if*

$$\phi(u, v) = \sum_{\lambda=1}^{\ell} f_\lambda(u, v) g_\lambda(u, v) w_\lambda \qquad \text{for all } u \in U, v \in V.$$

[1] For a finite dimensional associative k-algebra A with unity, the *radical* $\operatorname{rad} A$ is the intersection of all maximal twosided ideals of A. An algebra A is called *semisimple* if $\operatorname{rad} A = \{0\}$, see [9,14] for more details.

2. *The length of a shortest quadratic computation for ϕ is called the multiplicative complexity of ϕ and is denoted by $C(\phi)$.*
3. *If A is a finite dimensional associative k-algebra with unity, then the multiplicative complexity of A is defined as the multiplicative complexity of the multiplication map of A, which is a bilinear map $A \times A \to A$, and is denoted by $C(A)$.*

If we require that $f_\lambda \in U^*$ and $g_\lambda \in V^*$ in the above Definition 1, we get *bilinear computations* and *bilinear complexity* (also called *rank*). We denote the bilinear complexity of a bilinear map ϕ by $R(\phi)$ and the bilinear complexity of an associative algebra A by $R(A)$. We have $C(\phi) \le R(\phi) \le 2 \cdot C(\phi)$ for any bilinear map ϕ. Except for trivial cases, the second inequality is always strict, see [12].

1.2 Previous Results

The best general lower bound for the multiplicative complexity of an associative algebra A is due to Alder and Strassen [1], they show

$$C(A) \ge 2 \dim A - t\,, \tag{1}$$

where t is the number of maximal twosided ideals in A. This has recently been improved for a large class of semisimple algebras by Bläser [6]: if A is semisimple and if in the decomposition of $A \cong A_1 \times \cdots \times A_t$ into simple factors $A_\tau = {D_\tau}^{n_\tau \times n_\tau}$ with division algebra D_τ, each A_τ is noncommutative, then

$$C(A) \ge \tfrac{5}{2} \dim A - 3(n_1 + \cdots + n_t). \tag{2}$$

Specifically, the multiplicative complexity of $n \times n$–matrix multiplication is at least $\frac{5}{2}n^2 - 3n$. While the lower bound (2) also holds for algebras with nonzero radical in the case of bilinear complexity [4,5] provided that $A/\operatorname{rad} A$ fulfils the above condition, examples are presented in [6, Sec. 6] that show that those methods do not transfer to the multiplicative complexity for algebras with nonzero radical.

1.3 New Results

The starting point of our work is the above observation that the results from [4, 5] for the bilinear complexity do not transfer to the multiplicative complexity if A has nonzero radical. As our main contribution, we improve the Alder-Strassen bound (1) for various classes of algebras with nonzero radical.

As our first main result, we obtain the lower bound

$$C(A) \ge \dim A + \dim(\operatorname{rad} A)^m + \dim(\operatorname{rad} A)^n - \dim(\operatorname{rad} A)^{n+m-1}.$$

for any $n, m > 0$ (Theorem 1). This bound works very well for algebras with "growing" radical, that means, $2\dim(\operatorname{rad} A)^m$ is much larger than $\dim \operatorname{rad} A$

where m is the smallest natural number such that $(\operatorname{rad} A)^{2m-1} = \{0\}$. As an example, we apply this bound to the algebras $k[X_1, \ldots, X_n]/\, I_{d+1}(X_1, \ldots, X_n)$ where $I_d(X_1, \ldots, X_n)$ denotes the ideal generated by all monomials of degree d in $X_1, \ldots, X_n$. Furthermore, we obtain a sequence of explicitly given algebras A_n with $C(A_n) \geq (3 - o(1)) \dim A_n$. To our knowledge, this is the best lower bound known for a concrete algebra. (For existential results of algebras of high complexity, see [7].)

More complicated is the case of algebras with "nongrowing" radical, the most important one is probably the algebra of upper triangular $n \times n$–matrices $\mathrm{T}_n(k)$. We deal with this in Section 5. Our second main result is Lemma 7. This lemma provides a way to prove lower bounds above the Alder Strassen bound for algebras A with "nongrowing" radical. Here, $C(A)$ is estimated from below by the multiplicative complexity of a bilinear map ψ obtained from the multiplication in A by restricting to some subspaces. After that, $C(\psi)$ can be estimated using techniques introduced in [6]. In Section 6, we apply this method to the algebra of upper triangular matrices and get the lower bound

$$C(\mathrm{T}_n(k)) \geq (2\tfrac{1}{8} - o(1)) \dim \mathrm{T}_n(k).$$

This is the first bound for $\mathrm{T}_n(k)$ significantly above the Alder–Strassen bound. Prior to this, we only knew that the Alder–Strassen bound could be improved by an additive amount of one for $n \geq 3$ [11], that is, $\mathrm{T}_n(k)$ is not an algebra of minimal rank.

2 Preliminaries

For the reader's convenience, we compile some preliminaries to which we will refer frequently in the subsequent sections. In the first part of this section, we present an alternative characterization of multiplicative complexity, the so-called "tensorial notion" and state the relevant results. In the second part, we briefly review the lower bound techniques used by Alder and Strassen to prove their lower bound. These techniques basically are sophisticated refinements of the substitution method due to Pan [13].

2.1 Characterizations of Multiplicative Complexity

In the previous section, we have introduced the multiplicative complexity of a bilinear map in terms of computations. A second useful characterization of multiplicative complexity is the so-called "tensorial notion" (see [8, Chap. 14.4] for the bilinear complexity). With a bilinear map $\phi : U \times V \to W$, we may associate a *coordinate tensor* (or tensor for short) which is basically a "three-dimensional matrix": we fix bases $u_1, \ldots, u_m$ of U, $v_1, \ldots, v_n$ of V, and $w_1, \ldots, w_p$ of W. There are unique scalars $t_{\mu,\nu,\rho} \in k$ such that $\phi(u_\mu, v_\nu) = \sum_{\rho=1}^{p} t_{\mu,\nu,\rho} w_\rho$ for all $1 \leq \mu \leq m$, $1 \leq \nu \leq n$. Then $t = (t_{\mu,\nu,\rho}) \in k^{m \times n \times p}$ is the tensor of ϕ (with respect to the chosen bases). On the other hand, any given tensor also defines

a bilinear map after choosing bases. We define the multiplicative complexity of the tensor t by $C(t) := C(\phi)$. In the same way, the bilinear complexity of t is $R(t) := R(\phi)$. (This is in both cases well-defined.)

With each tensor $t = (t_{\mu,\nu,\rho})$, we may associate three sets of matrices, the *slices* of t. The matrices $Q_\mu = (t_{\mu,\nu,\rho})_{1\le\nu\le n,1\le\rho\le p} \in k^{n\times p}$ with $1 \le \mu \le m$ are called the 1-slices of t, the matrices $S_\nu = (t_{\mu,\nu,\rho})_{1\le\mu\le m,1\le\rho\le p} \in k^{m\times p}$ with $1 \le \nu \le n$ the 2-slices, and finally $T_\rho = (t_{\mu,\nu,\rho})_{1\le\mu\le m,1\le\nu\le n} \in k^{m\times n}$ with $1 \le \rho \le p$ are called the 3-slices of t. When dealing with bilinear complexity, it makes no difference which of the three sets of slices we consider. In the case of multiplicative complexity, however, the 3-slices play a distinguished role.

Lemma 1. *Let k be a field and t be a tensor with 3-slices $T_1, \dots, T_p \in k^{m\times n}$. Then $C(t) \le \ell$ if and only if there are (column) vectors $u_\lambda, v_\lambda \in k^{m+n}$ for $1 \le \lambda \le \ell$ such that with $P_\lambda := u_\lambda \cdot v_\lambda^\top \in k^{(m+n)\times(m+n)}$*

$$\begin{pmatrix} 0 & T_1 \\ T_1^\top & 0 \end{pmatrix}, \dots, \begin{pmatrix} 0 & T_p \\ T_p^\top & 0 \end{pmatrix} \in \operatorname{lin}\{P_1 + P_1^\top, \dots, P_\ell + P_\ell^\top\}\,. \tag{3}$$

Here, $T^\top$ denotes the transpose of a matrix T and $\operatorname{lin}\{\dots\}$ denotes the linear span. A proof of this lemma is straight forward (see for instance [12, Thm. 3.2]).

If $T_1, \dots, T_p$ are the 3-slices of a tensor t, we will occasionally write $C(T_1, \dots, T_p)$ instead of $C(t)$ and $R(T_1, \dots, T_p)$ instead of $R(t)$. By multiplying (3) with

$$\begin{pmatrix} X & 0 \\ 0 & Y^\top \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} X^\top & 0 \\ 0 & Y \end{pmatrix}$$

from the left and right, respectively, it follows from Lemma 1 that if $X \in k^{m\times m}$ and $Y \in k^{n\times n}$ are invertible matrices, then

$$C(T_1, \dots, T_p) = C(X \cdot T_1 \cdot Y, \dots, X \cdot T_p \cdot Y)\,. \tag{4}$$

2.2 The Lower Bound Techniques of Alder and Strassen

Beside the original paper of Alder and Strassen, [10, Chap. IV.2] and [8, Chap. 17] are excellent treatments of the results of Alder and Strassen. We have taken the term "separate" and the extension lemma from there, but everything is also contained in the work of Alder and Strassen [1].

Definition 2. *Let U, V, W be vector spaces and $\beta = (f_1, g_1, w_1, \dots, f_\ell, g_\ell, w_\ell)$ be a quadratic computation for a bilinear map $\phi : U \times V \to W$. Let $U_1 \subseteq U$, $V_1 \subseteq V$, and $W_1 \subseteq W$ be subspaces. The computation β* separates *(U_1, V_1, W_1), if there is a set of indices $I \subseteq \{\lambda \mid w_\lambda \notin W_1\}$ such that after possibly exchanging some of the f_λ with the corresponding g_λ, we have $(U_1 \times V_1) \cap \bigcap_{i\in I} \ker f_i = \{0\}$.*

The latter condition is equivalent to the condition that $(f_i|_{U_1\times V_1})_{i\in I}$ generate the dual space $(U_1 \times V_1)^*$.

If $\phi : U \times V \to W$ is a bilinear map and $U_1 \subseteq U$ and $V_1 \subseteq V$ are subspaces, then $(u+U_1, v+V_1) \mapsto \phi(u,v)+\tilde{W}$ defines a bilinear map $U/U_1 \times V/V_1 \to W/\tilde{W}$ where $\tilde{W} := \operatorname{lin}\{\phi(U_1, V)\} + \operatorname{lin}\{\phi(U, V_1)\}$. This map is called the *quotient* of ϕ by U_1 and V_1 and is denoted by $\phi/(U_1 \times V_1)$. We have the following lower bound. (See [8, Lem. 17.17] where also a proof is given.)

Lemma 2. *Let U, V, and W be vector spaces and $\beta = (f_1, g_1, w_1, \ldots, f_\ell, g_\ell, w_\ell)$ be a quadratic computation for some bilinear map $\phi : U \times V \to W$. Let $U_1 \subseteq U$, $V_1 \subseteq V$, and $W_1 \subseteq W$ be subspaces such that β separates (U_1, V_1, W_1). Let π be an endomorphism of W such that $W_1 \subseteq \ker \pi$. Then*

$$\ell \geq C((\pi \circ \phi)/(U_1 \times V_1)) + \dim U_1 + \dim V_1 + \#\{\lambda \mid w_\lambda \in W_1\}.$$

If ϕ is the multiplication map of an associative algebra A and I is a twosided ideal of A, then $\phi/(I \times I)$ is the multiplication map of the quotient algebra A/I.

Corollary 1. *Let A be an algebra and $I \subseteq A$ a twosided ideal. Let β be a quadratic computation for A of length ℓ that separates $(I, I, \{0\})$. Then $\ell \geq C(A/I) + 2 \dim I$.*

To achieve good lower bounds by means of Lemma 2, one has to find an optimal bilinear computation which separates a "large" triple. An important tool to solve this task is the following "extension lemma", see [8, Lem. 17.18].

Lemma 3 (Alder and Strassen). *Let U, V, W be vector spaces and β be a quadratic computation for a bilinear map $\phi : U \times V \to W$. Let $U_1 \subseteq U_2 \subseteq U$, $V_1 \subseteq V$, and $W_1 \subseteq W$ be subspaces such that β separates (U_1, V_1, W_1). Then β separates also (U_2, V_1, W_1), or there is some $u \in U_2 \setminus U_1$ such that $\phi(u, V) \subseteq \operatorname{lin}\{\phi(U_1, V_1)\} + W_1$.*

In the course of their proof, Alder and Strassen first deal with the radical of an algebra A and then turn to the semisimple quotient algebra $A/\operatorname{rad} A$. The following lemma contains the first of the two important statements established by Alder and Strassen, see [1, Lem. 2] or [8, Prop. 17.20].

Lemma 4 (Alder and Strassen). *Let β be a quadratic computation for an associative algebra A. Then β separates $(\operatorname{rad} A, \operatorname{rad} A, \{0\})$.*

3 Lower Bounds: "Growing" Radicals

In this section, we develop a method that works very well if the radical of an algebra A "grows", more precisely, $2 \dim(\operatorname{rad} A)^m$ is much larger than $\dim \operatorname{rad} A$ where m is the smallest natural number such that $(\operatorname{rad} A)^{2m-1} = \{0\}$. This method has already been applied to the bilinear complexity, see [4,5]. In contrast to the "hard" case (in Section 5), we do not loose anything when extending this method to the multiplicative complexity.

We build upon the following lower bound which is given in [2, Lem. 2].

Lemma 5. *Let A be an associative algebra. If $U, V \subseteq \operatorname{rad} A$ are vector spaces, then*

$$C(A) \geq \dim U + \dim V + \min_{\substack{X \oplus (U \times V) \\ = A \times A}} \{\dim \operatorname{lin}\{xx' \mid (x, x') \in X\}\}.$$

To put this lemma into effective use, we have to estimate the dimension of the subspaces $\operatorname{lin}\{xx' \mid (x, x') \in X\}$ in the above Lemma 5. This is done through the following lemma.

Lemma 6. *Let A be an associative algebra and $m, n > 0$ be natural numbers. For any vector space X such that $X \oplus ((\operatorname{rad} A)^m \times (\operatorname{rad} A)^n) = A \times A$,*

$$\operatorname{lin}\{xx' \mid (x, x') \in X\} + (\operatorname{rad} A)^{m+n-1} = A.$$

Proof. For the ease of notation, if $Y \subseteq A \times A$ is a vector space, let $\hat{Y}$ denote the vector space $\operatorname{lin}\{yy' \mid (y, y') \in Y\}$.

For $0 \leq \mu < m$ and $0 \leq \nu < n$, let $X_{\mu,\nu} \subseteq X$ be a vector space such that $X_{\mu,\nu} \oplus ((\operatorname{rad} A)^m \times (\operatorname{rad} A)^n) = (\operatorname{rad} A)^\mu \times (\operatorname{rad} A)^\nu$. Such an $X_{\mu,\nu}$ exists, because $X \oplus ((\operatorname{rad} A)^m \times (\operatorname{rad} A)^n) = A \times A$. As $X \cap ((\operatorname{rad} A)^m \times (\operatorname{rad} A)^n) = \{0\}$, it is also unique. Furthermore, $X_{\mu,\nu} \subseteq X_{\mu',\nu'}$ for $\mu' \leq \mu$ and $\nu' \leq \nu$.

For any $(u, v) \in (\operatorname{rad} A)^\mu \times (\operatorname{rad} A)^\nu$, there are $(a, b) \in (\operatorname{rad} A)^m \times (\operatorname{rad} A)^n$ such that $(u, v) + (a, b) \in X_{\mu,\nu}$. Thus $(u + a)(v + b) \in \hat{X}_{\mu,\nu}$. But $(u + a)(v + b) \in uv + (\operatorname{rad} A)^{\mu+\nu+1}$. Letting (u, v) run through all elements of $(\operatorname{rad} A)^\mu \times (\operatorname{rad} A)^\nu$, we get

$$\hat{X}_{\mu,\nu} + (\operatorname{rad} A)^{\mu+\nu+1} = (\operatorname{rad} A)^{\mu+\nu}. \tag{5}$$

We now prove by backward induction in $\mu + \nu$ that

$$\hat{X}_{\mu,\nu} + (\operatorname{rad} A)^{m+n-1} = (\operatorname{rad} A)^{\mu+\nu} \qquad \text{for all } \mu < m,\ \nu < n.$$

For $\mu = \nu = 0$, this is the claim of the lemma.

The induction start ($\mu = m - 1$, $\nu = n - 1$) follows directly from (5). For the induction step, let μ and ν be given such that $\mu + \nu < m + n - 2$. We assume that $\mu < m - 1$, the case $\nu < n - 1$ follows completely alike. By substituting the induction hypothesis $\hat{X}_{\mu+1,\nu} + (\operatorname{rad} A)^{m+n-1} = (\operatorname{rad} A)^{\mu+\nu+1}$ into (5), we obtain $\hat{X}_{\mu,\nu} + \hat{X}_{\mu+1,\nu} + (\operatorname{rad} A)^{m+n-1} = (\operatorname{rad} A)^{\mu+\nu}$. Now the claim follows from the fact that $\hat{U} \subseteq \hat{V}$ if $U \subseteq V$. □

Combining the last lemma with Lemma 5 shows the following lower bound.

Theorem 1. *Let A be a an associative algebra. For all $m, n > 0$,*

$$C(A) \geq \dim A + \dim(\operatorname{rad} A)^m + \dim(\operatorname{rad} A)^n - \dim(\operatorname{rad} A)^{n+m-1}.$$

4 Multiplying Multivariate Power Series

For indeterminates $X_1, \ldots, X_n$, let $I_d(X_1, \ldots, X_n)$ denote the ideal generated by all monomials of degree d. Let $P_{n,d}$ be the algebra $k[X_1, \ldots, X_n]/\, I_{d+1}(X_1, \ldots, X_n)$. The multiplication in $P_{n,d}$ can be interpreted as the multiplication of n–variate power series where we only compute with the coefficients of the monomials with degree at most d. For the algebras $P_{n,d}$, the methods of the preceding section give a nice lower bound. The below theorem follows at once from Theorem 1 and the fact that the dimension of $\dim P_{n,d}$ is $\binom{n+d}{n}$.

Theorem 2. *For any $n > 0$ and $d > 0$,*

$$C(P_{n,d}) \geq 3 \cdot \tbinom{n+d}{n} - \tbinom{n+\lceil d/2 \rceil}{n} - \tbinom{n+\lfloor d/2 \rfloor}{n}.$$

Remark 1. If we keep $d > 1$ fixed, then $C(P_{n,d}) \geq (3 - o(1)) \dim P_{n,d}$ (as a function in n). This gives a sequence of explicitly given algebras such that the multiplicative complexity comes arbitrarily close to three times the dimension of the algebra. This is the the best we can expect from currently known lower bounds techniques for bilinear problems. To our knowledge, this is the first lower bound of this kind.

5 Lower Bounds: The Hard Case

Throughout the remainder of this section, we use the following notations: as usual, A is an associative algebra. We denote its multiplication map by ϕ. We assume that we have a decomposition $A = I \oplus X \oplus Y$ (as vector spaces) with vector spaces X and Y and a twosided ideal I. Furthermore, $I^2 = \{0\}$ and $Y \cdot I = \{0\}$. Moreover, let $U \subseteq X$ and $V \subseteq Y$ be vector spaces such that for all projections π of A onto $I \oplus U \oplus V$, the bilinear map $\pi \circ \phi$ is *1–concise*, that is, its left kernel $\{a \in A \mid \pi \circ \phi(a, A) = \{0\}\}$ equals $\{0\}$.

Our general plan looks as follows: using Lemma 2, we reduce the proof of a lower bound for $C(\phi)$ to the proof of a lower bound for $C(\phi|_{X \times I})$ where $\phi|_{X \times I}$ denotes the restriction of ϕ to $X \times I$. This reduction works for any algebra with the above decomposition property. After that, we have to estimate $C(\phi|_{X \times I})$ individually. This is done in the next section for the particularly important case of upper triangular matrices. The main result of this section is the following lower bound.

Lemma 7. *With the notations from above,*

$$C(A) \geq C(\phi|_{X \times I}) + \dim A + \dim Y - \dim U - \dim V.$$

Proof. Let $\beta = (f_1, g_1, w_1, \ldots, f_\ell, g_\ell, w_\ell)$ be a quadratic computation for A. Since $w_1, \ldots, w_\ell$ generate A, we may assume that $\operatorname{lin}\{w_{\ell-m+1}, \ldots, w_\ell\} \oplus I \oplus U \oplus V = A$ where $m = \dim A - \dim I - \dim U - \dim V$. Let π denote the projection along

$\lin\{w_{\ell-m+1},\dots,w_\ell\}$ onto $I\oplus U\oplus V$. Then $\beta'=(f_1,g_1,w'_1,\dots,f_{\ell'},g_{\ell'},w'_{\ell'})$ with $w'_\lambda=\pi(w_\lambda)$ and $\ell'=\ell-m$ is a quadratic computation for $\pi\circ\phi$.

We claim that β' separates $(A,\{0\},\{0\})$. If this was not the case, then there would be an $a\in A\setminus\{0\}$ by Lemma 3 such that

$$\pi\circ\phi(a,A)\subseteq\phi(\{0\},\{0\})+\{0\}=\{0\}$$

contradicting the assumption that $\pi\circ\phi$ is 1–concise.

From the definition of "separate", it follows that β' also separates $(I\oplus Y,\{0\},\{0\})$. In other words, $f_1|_{(I\oplus Y)\times\{0\}},\dots,f_{\ell'}|_{(I\oplus Y)\times\{0\}}$ generate $((I\oplus Y)\times\{0\})^*$ after possibly exchanging some of the f_λ with the corresponding g_λ.

Let $\psi=(\pi\circ\phi)|_{A\times I}$. Obviously $\hat\beta=(\hat f_1,\hat g_1,w'_1,\dots,\hat f_{\ell'},\hat g_{\ell'},w'_{\ell'})$ with $\hat f_\lambda=f_\lambda|_{A\times I}$ and $\hat g_\lambda=g_\lambda|_{A\times I}$ is a quadratic computation for ψ.

As $(I\oplus Y)\times\{0\}\subseteq A\times I$, $\hat f_\lambda|_{(I\oplus Y)\times\{0\}}=(f_\lambda|_{A\times I})|_{(I\oplus Y)\times\{0\}}=f_\lambda|_{(I\oplus Y)\times\{0\}}$. From this, we get that also $\hat\beta$ separates $(I\oplus Y,\{0\},\{0\})$. Lemma 2 now yields the lower bound

$$\ell'\geq C(\psi/(I\oplus Y)\times\{0\})+\dim I+\dim Y. \tag{6}$$

By the definition of "quotient" in Section 2.2, $\psi/(I\oplus Y)\times\{0\}$ is a bilinear map $A/(I\oplus Y)\times I\to(I\oplus U\oplus V)/\tilde W$ that maps $(a+(I\oplus Y),b)$ to $\pi\circ\phi(a,b)+\tilde W$ where $\tilde W=\lin\{\pi\circ\phi(A,\{0\})\}+\lin\{\pi\circ\phi(I\oplus Y,I)\}$. (For a vector space Z, we identify $Z/\{0\}$ with Z.) Since $I^2=\{0\}$ and $Y\cdot I=\{0\}$ by assumption, $\tilde W=\{0\}$. For $x+I\oplus Y\in A/(I\oplus Y)$ and $t\in I$,

$$\psi/(I\oplus Y)\times\{0\}(x+I\oplus Y,t)=\pi\circ\phi(x,t)=xt,$$

since $x\cdot t\in I$. Thus, the following diagram commutes

$$\begin{array}{ccc} A/(I\oplus Y)\times I & \xrightarrow{\psi/(I\oplus Y)\times\{0\}} & I\oplus U\oplus V \\ {\scriptstyle h\times id}\downarrow & & \uparrow \\ X\times I & \xrightarrow{\phi} & I \end{array}$$

where $h:A/(I\oplus Y)\to X$ denotes the canonical isomorphism. Hence we obtain $C(\psi/(I\oplus Y)\times\{0\})\geq C(\phi|_{X\times I})$.

Exploiting $\ell'=\ell-m$ and choosing β to be an optimal computation, the claim of the lemma follows from (6). □

6 Multiplication of Upper Triangular Matrices

We now apply the results of the preceding section to the algebra $\mathrm{T}_n(k)$ of upper triangular $n\times n$–matrices with entries from k. For the sake of simplicity, we assume that n is even.

In the following, let $e_{i,j}\in\mathrm{T}_n(k)$ denote the matrix that has a one in position (i,j) and zeros elsewhere for $1\leq i\leq j\leq n$. The radical R of $\mathrm{T}_n(k)$ equals

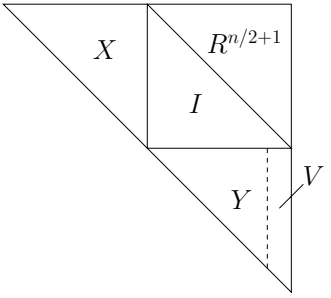


Fig. 1. The decomposition of $\mathrm{T}_n(k)$

the linear span of all $e_{i,j}$ with $i < j$, that is, R is the set of all matrices with purely zeros on the diagonal. More general, the power R^h equals the linear span of all $e_{i,j}$ with $i + h \leq j$.

We will first step down from $\mathrm{T}_n(k)$ to the quotient $A = \mathrm{T}_n(k)/R^{n/2+1}$. By Corollary 1, we obtain

$$C(\mathrm{T}_n(k)) \geq C(A) + 2 \dim R^{n/2+1} = C(A) + \tfrac{1}{4}n^2 - \tfrac{1}{2}n. \tag{7}$$

The multiplication in A corresponds to the multiplication of upper triangular matrices where we do not compute the entries in the positions (i, j) with $i + n/2 + 1 \leq j$. We use this representation in the following.

Next, we have to instantiate I, X, Y, U, and V. For the remainder of this section, let $m = n/2$. We choose (see Figure 1 for an illustration)

$$\begin{aligned} I &= \mathrm{lin}\{e_{i,j} \mid i \leq m \text{ and } j > m\}, & X &= \mathrm{lin}\{e_{i,j} \mid i \leq m \text{ and } j \leq m\}, \\ Y &= \mathrm{lin}\{e_{i,j} \mid i > m \text{ and } j > m\}, & U &= \{0\}, \\ V &= \mathrm{lin}\{e_{m+1,n}, e_{m+2,n}, \ldots, e_{n,n}\}. \end{aligned}$$

Obviously, $A = I \oplus X \oplus Y$. A straightforward calculation shows that $I^2 = \{0\}$ and $Y \cdot I = \{0\}$ (in A). Moreover, to fulfil the assumptions of Lemma 7, we have to show that for any projection π onto $I \oplus U \oplus V = I \oplus V$, $\pi \circ \phi$ is 1–concise, where ϕ denotes the multiplication map of A. So for each $e_{i,j} \in A$, we have to find an element $a \in A$ such that $\pi(e_{i,j} \cdot a) \neq 0$. We consider three cases: if $e_{i,j} \in I$, that is, $i \leq m$ and $j > m$, then $e_{i,j} \cdot e_{j,j} = e_{i,j} \in I$, thus $\pi(e_{i,j}) = e_{i,j} \neq 0$. If $e_{i,j} \in X$, i.e., $i \leq m$ and $j \leq m$, then $e_{i,j} \cdot e_{j,m+1} = e_{i,m+1} \in I$. If finally $e_{i,j} \in Y$, that is, $i > m$ and $j > m$, then $e_{i,j} \cdot e_{j,n} = e_{j,n} \in V$, thus $\pi(e_{i,j} \cdot e_{j,n}) = e_{j,n} \neq 0$. The 1-conciseness of $\pi \circ \phi$ follows from this.

It remains to estimate $C(\phi|_{X \times I})$. Our aim is to use the following lemma which is proven in [6, Lemma 4]. In what follows, $[B, C] := BC - CB$ denotes the *Lie product* of two matrices.

Lemma 8. *Let k be a field. Let t be a tensor with 3-slices $I_N, B, C \in k^{N \times N}$. Then $C(t) \geq N + \frac{1}{2} \operatorname{rk}[B, C]$.*

To utilize this lemma, we have to determine the tensor of $\phi|_{X\times I}$: for a clearer presentation, we choose the basis

$$\underbrace{e_{1,1},\dots,e_{1,m}}_{\text{first group}},\ \dots,\ \underbrace{e_{i,i},\dots,e_{i,m}}_{i\text{th group}},\ \dots,\quad \underbrace{e_{m,m}}_{m\text{th group}}$$

for X (in this row-wise order) and the basis

$$\underbrace{e_{1,m+1},\dots,e_{m,m+1}}_{\text{first group}},\ \dots,\ \underbrace{e_{j,m+j},\dots,e_{m,m+j}}_{j\text{th group}},\ \dots,\quad \underbrace{e_{m,n}}_{m\text{th group}}$$

for I (column-wise ordering). The third basis (again for I, since $\Phi|_{X\times I}$ is a mapping $X\times I\to I$) equals the second basis (but we forget about the groups). We denote the 3–slice of the tensor of $\phi|_{X\times I}$ that corresponds to $e_{i,m+j}$ by $T_{i,j}$ for $j\le i\le m$. The $T_{i,j}$ are matrices of size $M\times M$ where $M=\frac{1}{2}m(m+1)$. We associate a block structure with the $T_{i,j}$'s induced by the above groups of the first and second basis.

In $T_{i,j}$, the only positions with nonzero entries are in the block at position (i,j), that is, in the positions whose rows and columns correspond to the vectors of the ith and jth group of the above two bases, respectively.

An easy calculation shows that the entries of $T_{i,j}$ within these positions equal

$$\begin{pmatrix} Z_{i-j,j} \\ I_j \end{pmatrix} \tag{8}$$

where I_κ denotes the $\kappa\times\kappa$–identity matrix and $Z_{\mu,\nu}$ denotes the zero matrix of size $\mu\times\nu$. In particular, the 3–slices are linearly independent. From a macroscopic point of view with respect to the above block structure, the $T_{i,j}$ are block lower triangular matrices "of the form $e_{i,j}$" with the above matrix (8) as the only nonzero entry (instead of a one).

By Lemma 1, the fact that $C(\phi|_{X\times I})\le\ell$ is equivalent to the existence of rank one matrices $P_1,\dots,P_\ell$ such that

$$\begin{pmatrix} 0 & T_{i,j} \\ T_{i,j}^\top & 0 \end{pmatrix} \in \operatorname{lin}\{P_1+P_1^\top,\dots,P_\ell+P_\ell^\top\} \qquad \text{for } i\le j\le m.$$

We now exploit the Steinitz exchange to save one product for each tensor $T_{\mu,\nu}$ with $\mu\ge\nu+2$: there are matrices $S_1,\dots,S_m$ and $Q_1,\dots,Q_{m-1}$ in $\operatorname{lin}\{T_{\mu,\nu}\mid \mu\ge\nu+2\}$ such that after a suitable permutation of the $P_1,\dots,P_\ell$

$$\begin{pmatrix} 0 & T_{\mu,\mu} \\ T_{\mu,\mu}^\top & 0 \end{pmatrix} - \begin{pmatrix} 0 & S_\mu \\ S_\mu^\top & 0 \end{pmatrix}, \begin{pmatrix} 0 & T_{\nu,\nu-1} \\ T_{\nu,\nu-1}^\top & 0 \end{pmatrix} - \begin{pmatrix} 0 & Q_\nu \\ Q_\nu^\top & 0 \end{pmatrix} \in$$
$$\operatorname{lin}\{P_1+P_1^\top,\dots,P_{\ell-s}+P_{\ell-s}^\top\} \quad \text{for } 1\le\mu\le m,\ 1\le\nu\le m-1, \tag{9}$$

where $s=\frac{1}{2}m(m+1)-m-(m-1)$. Thus we have killed s products.

Let $\lambda_1, \ldots, \lambda_m \in k$ be pairwise distinct. Define

$$\begin{aligned} E &= T_{1,1} - S_1 + \cdots + T_{m,m} - S_m, \\ B &= \lambda_1(T_{1,1} - S_1) + \cdots + \lambda_m(T_{m,m} - S_m), \\ C &= T_{2,1} - Q_1 + \cdots + T_{m,m-1} - Q_{m-1}. \end{aligned}$$

From (9), we obtain

$$C(\phi|_{X \times I}) \geq s + C(E, B, C). \tag{10}$$

With respect to the above block structure, E has solely identity matrices on the main diagonal and zero matrices one the first subdiagonal. The matrix B has λ_μ multiples of identity matrices on the main diagonal and also zero matrices on the first subdiagonal. The matrix C has zero matrices on the diagonal and "nearly" identity matrices, more precisely, a line of zeros with an identity matrix (as depicted in (8)) one the first subdiagonal.

The matrix E is invertible. By (4),

$$C(E, B, C) = C(I_M, BE^{-1}, CE^{-1}). \tag{11}$$

Due to the structure of E, E^{-1} also has solely identity matrices on the main diagonal and zero matrices on the first subdiagonal. Thus, BE^{-1} has λ_μ multiples of identity matrices on the main diagonal and zero matrices on the first subdiagonal. In the same way, CE^{-1} has zero matrices on the diagonal and "nearly" identity matrices on the first subdiagonal. Some easy algebra shows that due to this structure, the Lie product $[BE^{-1}, CE^{-1}]$ has zero matrices in the blocks on the main diagonal and the matrix

$$\underbrace{(\lambda_{j+1} - \lambda_j)}_{\neq 0} \cdot \begin{pmatrix} Z_{1,j} \\ I_j \end{pmatrix}$$

in the $(j+1, j)$–block (on the first subdiagonal) for $1 \leq j \leq m-1$. Hence, the rank of $[BE^{-1}, CE^{-1}]$ is at least $1 + 2 + \cdots + m - 1 = \frac{1}{2}(m-1)m$.

Together with (10), (11), and Lemma 8, the last statement implies the following lower bound.

Lemma 9. *With the notations from above,* $C(\phi|_{X \times I}) \geq \frac{5}{4}m^2 - \frac{5}{4}m + 1$.

Exploiting (7) and then bounding $C(A)$ by Lemma 7 and Lemma 9, we obtain the following lower bound.

Theorem 3. *For even n, the multiplicative complexity of the multiplication of upper triangular matrices of size $n \times n$ has the lower bound*

$$C(\mathrm{T}_n(k)) \geq \tfrac{17}{16}n^2 - \tfrac{3}{8}n + 1 \geq (2\tfrac{1}{8} - o(1)) \dim \mathrm{T}_n(k).$$

Remark 2. For odd n, the same approach also yields $(2\frac{1}{8} - o(1)) \dim \mathrm{T}_n(k)$ as a lower bound. A quick solution goes a follows: simply embed $\mathrm{T}_{n-1}(k)$ into $\mathrm{T}_n(k)$ and apply the above theorem. We only loose an additive amount of $O(n) \leq o(\dim \mathrm{T}_n(k))$.

References

1. A. Alder and V. Strassen. On the algorithmic complexity of associative algebras. *Theoret. Comput. Sci.*, 15:201–211, 1981.
2. Markus Bläser. Bivariate polynomial multiplication. In *Proc. 39th Ann. IEEE Symp. on Foundations of Comput. Sci. (FOCS)*, pages 186–191, 1998.
3. Markus Bläser. Lower bounds for the multiplicative complexity of matrix multiplication. *Comput. Complexity*, 8:203–226, 1999.
4. Markus Bläser. *Untere Schranken für den Rang assoziativer Algebren*. Dissertation, Universität Bonn, 1999.
5. Markus Bläser. Lower bounds for the bilinear complexity of associative algebras. *Comput. Complexity*, 9:73–112, 2000.
6. Markus Bläser. A $\frac{5}{2}n^2$–lower bound for the multiplicative complexity of $n\times n$–matrix multiplication. In *Proc. 18th Int. GI–MIMD Symp. on Theoret. Aspects of Comput. Sci. (STACS)*, Lecture Notes in Comput. Sci. 2010, pages 99–110, 2001.
7. Werner Büchi. *Über eine Klasse von Algebren minimalen Rangs*. Dissertation, Universität Zürich, 1984.
8. Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic Complexity Theory*. Springer, 1997.
9. Yurij A. Drozd and Vladimir V. Kirichenko. *Finite Dimensional Algebras*. Springer, 1994.
10. Hans F. de Groote. *Lectures on the Complexity of Bilinear Problems*. Lecture Notes in Comput. Science 245. Springer, 1986.
11. Joos Heintz and Jacques Morgenstern. On associative algebras of minimal rank. In *Proc. 2nd Applied Algebra and Error Correcting Codes Conf. (AAECC)*, Lecture Notes in Comput. Sci. 228, pages 1–24. Springer, 1986.
12. Joseph Ja'Ja'. On the complexity of bilinear forms with commutativity. *SIAM J. Comput.*, 9:717–738, 1980.
13. Victor Ya. Pan. Methods for computing values of polynomials. *Russ. Math. Surv.*, 21:105–136, 1966.
14. Richard S. Pierce. *Associative Algebras*. Springer, 1982.
15. Volker Strassen. Vermeidung von Divisionen. *J. Reine Angew. Math.*, 264:184–202, 1973.
16. Volker Strassen. Algebraic complexity theory. In J. van Leeuven, editor, *Handbook of Theoretical Computer Science Vol. A*, pages 634–672. Elsevier Science Publishers B.V., 1990.

On Generating All Minimal Integer Solutions for a Monotone System of Linear Inequalities*

E. Boros[1], K. Elbassioni[2], V. Gurvich[1], L. Khachiyan[2], and K. Makino[3]

[1] RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway NJ 08854-8003; {boros,gurvich}@rutcor.rutgers.edu

[2] Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway NJ 08854-8003; elbassio@paul.rutgers.edu,leonid@cs.rutgers.edu

[3] Division of Systems Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka, 560-8531, Japan; makino@sys.es.osaka-u.ac.jp

Abstract. We consider the problem of enumerating all minimal integer solutions of a monotone system of linear inequalities. We first show that for any monotone system of r linear inequalities in n variables, the number of maximal infeasible integer vectors is at most rn times the number of minimal integer solutions to the system. This bound is accurate up to a $polylog(r)$ factor and leads to a polynomial-time reduction of the enumeration problem to a natural generalization of the well-known dualization problem for hypergraphs, in which dual pairs of hypergraphs are replaced by dual collections of integer vectors in a box. We provide a quasi-polynomial algorithm for the latter dualization problem. These results imply, in particular, that the problem of incrementally generating minimal integer solutions of a monotone system of linear inequalities can be done in quasi-polynomial time.

Keywords: Integer programming, complexity of incremental algorithms, dualization, quasi-polynomial time, monotone discrete binary functions, monotone inequalities, regular discrete functions.

1 Introduction

Consider a system of r linear inequalities in n integer variables

$$Ax \geq b, \quad x \in \mathcal{C} = \{x \in \mathbb{Z}^n \mid 0 \leq x \leq c\}, \tag{1}$$

where A is a rational $r \times n$-matrix, b is a rational r-vector, and c is a non-negative integral n-vector some or all of whose components may be infinite. We assume that (1) is a monotone system of inequalities: if $x \in \mathcal{C}$ satisfies (1) then any vector $y \in \mathcal{C}$ such that $y \geq x$ is also feasible. For instance, (1) is monotone if the

* The research of the first and third authors was supported in part by the Office of Naval Research (Grant N00014-92-J-1375), and the National Science Foundation (Grant DMS 98-06389). The research of the third and forth authors was supported in part by the National Science Foundation (Grant CCR-9618796).

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 92–103, 2001.

matrix A is non-negative. Let us denote by $\mathcal{F} = \mathcal{F}_{A,b,c}$ the set of all minimal feasible integral vectors for (1), i.e. $y \in \mathcal{F}$ if there is no solution x of (1) such that $x \leq y,\ x \neq y$. In particular, we have $\{x \in \mathcal{C} \mid Ax \geq b\} = \bigcup_{y \in \mathcal{F}} \{\ x \in \mathcal{C} \mid x \geq y\}$. In this paper, we are concerned with the problem of incrementally generating $\mathcal{F}$:

$GEN(\mathcal{F}_{A,b,c}, \mathcal{X})$: Given a monotone system (1) and a set $\mathcal{X} \subseteq \mathcal{F}_{A,b,c}$ of minimal feasible vectors for (1), either find a new minimal integral vector satisfying (1), or show that $\mathcal{X} = \mathcal{F}_{A,b,c}$.

The entire set $\mathcal{F} = \mathcal{F}_{A,b,c}$ can be constructed by initializing $\mathcal{X} = \emptyset$ and iteratively solving the above problem $|\mathcal{F}| + 1$ times.

If A is a binary matrix, and b, c are vectors of all ones, then $\mathcal{F}$ is the set of (characteristic vectors of) all minimal transversals to the hypergraph defined by the rows of A. In this case, problem $GEN(\mathcal{F}_{A,b,c}, \mathcal{X})$ turns into the well-known *hypergraph dualization problem*: incrementally enumerate all the minimal transversals (equivalently, all the maximal independent sets) for a given hypergraph (see, e.g., [3,10]). Some applications of the hypergraph dualization problem are discussed in [1,7,9]. The case where A is binary, c is the vector of all ones and b is arbitrary, is equivalent with the generation of so-called *multiple* transversals [5]. If A is integral and $c = +\infty$, the generation of $\mathcal{F}$ can also be regarded as the computation of the Hilbert basis for the ideal $\{x \in \mathbb{Z}^n \mid Ax \geq b,\ x \geq 0\}$. One more application of problem $GEN(\mathcal{F}_{A,b,c}, \mathcal{X})$ is related to stochastic programming, more precisely to the generation of minimal *p-efficient* points for a given probability distribution of a discrete random variable $\xi \in \mathbb{Z}^n$. An integer vector $y \in \mathbb{Z}^n$ is called p-efficient, if $Prob(\xi \leq y) \geq p$. It is known that for every probability distribution and every $p > 0$ there are finitely many minimal p-efficient points, furthermore that for r-concave probability distributions these points are exactly the minimal integral points of a corresponding convex monotone system (see, e.g., [14]).

Let $J^* = \{j \mid c_j = \infty\}$ and $J_* = \{1, \ldots, n\} \setminus J^*$ be, respectively, the sets of unbounded and bounded integer variables in (1). Consider an arbitrary vector $x = (x_1, \ldots, x_n) \in \mathcal{F}_{A,b,c}$ such that $x_j > 0$ for some $j \in J^*$. Then it is easy to see that

$$x_j \leq \max_{i:a_{ij}>0} \left\lceil \frac{b_i - \sum_{k \in J_*} \min\{0, a_{ik}\} c_k}{a_{ij}} \right\rceil < +\infty. \tag{2}$$

Since the bounds of (2) are easy to compute, and since appending these bounds to (1) does not change the set $\mathcal{F}_{A,b,c}$, we shall assume in the sequel that all components of the non-negative vector c are finite, even though this may not be the case for the original system. This assumption does not entail any loss of generality and allows us to consider $\mathcal{F}_{A,b,c}$ as a system of integral vectors in a finite box. We shall also assume that the input monotone system (1) is feasible, i.e., $\mathcal{F}_{A,b,c} \neq \emptyset$. For a finite and non-negative c this is equivalent to $Ac \geq b$.

Let $\mathcal{A}$ be a collection of integral vectors in $\mathcal{C}$ and let $\mathcal{A}^+ = \{x \in \mathcal{C} \mid x \geq a \text{ for some } a \in \mathcal{A}\}$ and $\mathcal{A}^- = \{x \in \mathcal{C} \mid x \leq a \text{ for some } a \in \mathcal{A}\}$ denote the ideal and filter generated by $\mathcal{A}$. Any element in $\mathcal{C} \setminus \mathcal{A}^+$ is called *independent of* $\mathcal{A}$. Let $\mathcal{I}(\mathcal{A})$ be the set of all maximal independent elements for $\mathcal{A}$, then for any finite box $\mathcal{C}$ we have the decomposition:

$$\mathcal{A}^+ \cap \mathcal{I}(\mathcal{A})^- = \emptyset, \quad \mathcal{A}^+ \cup \mathcal{I}(\mathcal{A})^- = \mathcal{C}. \tag{3}$$

In particular, if $\mathcal{A}$ is the set $\mathcal{F} = \mathcal{F}_{A,b,c}$ of all minimal feasible integral vectors for (1), then the ideal $\mathcal{F}^+$ is the solution set of (1), while the filter $\mathcal{C} \setminus \mathcal{F}^+$ is generated by the set $\mathcal{I}(\mathcal{F})$ of all maximal infeasible integral vectors for (1):

$$\{x \in \mathcal{C} \mid Ax \not\geq b\} = \bigcup_{y \in \mathcal{I}(\mathcal{F})} \{y\}^-.$$

It is known that the problem of incrementally generating all maximal infeasible vectors for (1) is NP-hard even if c is the vector of all ones and the matrix A is binary:

Proposition 1 (c.f. [12]). *Given a binary matrix A and a set $\mathcal{X} \subseteq \mathcal{I}(\mathcal{F}_{A,b,c})$ of maximal infeasible Boolean vectors for $Ax \geq b$, $x \in \{0,1\}^n$, it is NP-complete to decide if the set $\mathcal{X}$ can be extended, that is if $\mathcal{I}(\mathcal{F}_{A,b,c}) \setminus \mathcal{X} \neq \emptyset$.*

In contrast to that, we show in this paper that the problem of incrementally generating all minimal feasible vectors for (1) is unlikely to be NP-hard.

Theorem 1. *Problem $GEN(\mathcal{F}_{A,b,c}, \mathcal{X})$ can be solved in quasi-polynomial time $poly(|input|) + t^{o(\log t)}$, where $t = \max\{n, r, |\mathcal{X}|\}$.*

It was conjectured in [11] that problem $GEN(\mathcal{F}_{A,b,c}, \mathcal{X})$ cannot be solved in polynomial time unless P=NP.

To prove Theorem 1, we first bound the number of maximal infeasible vectors for (1) in terms of the dimension of the system and the number of minimal feasible vectors.

Theorem 2. *Suppose that the monotone system (1) is feasible, i.e., $Ac \geq b$. Then for any non-empty set $\mathcal{X} \subseteq \mathcal{F}_{A,b,c}$ we have*

$$|\mathcal{I}(\mathcal{X}) \cap \mathcal{I}(\mathcal{F}_{A,b,c})| \leq r \sum_{x \in \mathcal{X}} p(x), \tag{4}$$

where $p(x)$ is the number of positive components of x. In particular,

$$|\mathcal{I}(\mathcal{X}) \cap \mathcal{I}(\mathcal{F}_{A,b,c})| \leq rn|\mathcal{X}|,$$

which for $\mathcal{X} = \mathcal{F}_{A,b,c}$ leads to the inequality $|\mathcal{I}(\mathcal{F}_{A,b,c})| \leq rn|\mathcal{F}_{A,b,c}|$.

It should be mentioned that the bounds of Theorem 2 are sharp for $r = 1$, e.g., for the inequality $x_1 + \ldots + x_n \geq n$. For large r, these bounds are accurate up to a factor poly-logarithmic in r. To see this, let $n = 2k$ and consider the monotone system of $r = 2^k$ inequalities of the form

$$x_{i_1} + x_{i_2} + \cdots + x_{i_k} \geq 1,\ i_1 \in \{1, 2\},\ i_2 \in \{3, 4\}, \ldots, i_k \in \{2k-1, 2k\},$$

where $x = (x_1, \ldots, x_n) \in \mathcal{C} = \{x \in \mathbb{Z}^n \mid 0 \leq x \leq c\}$. For any positive integral vector c, this system has 2^k maximal infeasible integral vectors and only k minimal feasible integral vectors, i.e.,

$$|\mathcal{I}(\mathcal{F}_{A,b,c})| = \frac{rn}{2(\log r)^2} |\mathcal{F}_{A,b,c}|.$$

Needless to say that in general, $|\mathcal{F}_{A,b,c}|$ cannot be bounded by a polynomial in r, n, and $|\mathcal{I}(\mathcal{F}_{A,b,c})|$. For instance, for $n = 2k$ the system of k inequalities $x_1 + x_2 \geq 1$, $x_3 + x_4 \geq 1, \ldots, x_{2k-1} + x_{2k} \geq 1$ has 2^k minimal feasible binary vectors and only k maximal infeasible binary vectors.

Let us add finally that if the number of inequalities in (1) is fixed, then $|\mathcal{F}_{A,b,c}|$ can also be polynomially bounded by $|\mathcal{I}(\mathcal{F}_{A,b,c})|$, and accordingly, the set of all maximal infeasible integer vectors for (1) can be generated in quasi-polynomial time. In other words, Proposition 1 cannot hold for $r = const$ unless any problem in NP can be solved in quasi-polynomial time. Furthermore, for systems with fixed number of non-zero coefficients per inequality and bounded box size, problem $GEN(\mathcal{F}_{A,b,c}, \mathcal{X})$ can be efficiently solved in parallel (see [4]).

We prove Theorem 2 in Section 2, and then use this theorem in the next section to reduce problem $GEN(\mathcal{F}_{A,b,c}, \mathcal{X})$ to a natural generalization of the hypergraph dualization problem. Our generalized dualization problem replaces hypergraphs by collections of integer vectors in a box.

Theorem 3. *$GEN(\mathcal{F}_{A,b,c}, \mathcal{X})$ is polynomial-time reducible to the following problem:*

$DUAL(\mathcal{C}, \mathcal{A}, \mathcal{B})$: Given an integral box $\mathcal{C}$, a family of vectors $\mathcal{A} \subseteq \mathcal{C}$, and a collection of maximal independent elements $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$, either find a new maximal independent element $x \in \mathcal{I}(\mathcal{A}) \setminus \mathcal{B}$, or prove that $\mathcal{B} = \mathcal{I}(\mathcal{A})$.

Note that for $\mathcal{C} = \{0, 1\}^n$, problem $DUAL(\mathcal{C}, \mathcal{A}, \mathcal{B})$ turns into the hypergraph dualization problem. Other applications of the dualization problem on boxes can be found in [2,6,13]. In Section 4 we extend the hypergraph dualization algorithm of [8] to problem $DUAL(\mathcal{C}, \mathcal{A}, \mathcal{B})$ and show that the latter problem can be solved in quasi-polynomial time:

Theorem 4. *Given two sets $\mathcal{A}$, and $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$ in an integral box $\mathcal{C} = \{x \in \mathbb{Z}^n \mid 0 \leq x \leq c\}$, problem $DUAL(\mathcal{C}, \mathcal{A}, \mathcal{B})$ can be solved in $poly(n, m) + m^{o(\log m)}$ time, where $m = |\mathcal{A}| + |\mathcal{B}|$.*

Clearly, Theorem 1 follows from Theorems 3 and 4. The special cases of Theorems 2 and 3 for Boolean systems $x \in \{0, 1\}^n$ can be found in [5].

The remainder of the paper consists of the proofs of Theorems 2, 3 and 4 in Sections 2, 3, 4 respectively.

2 Bounding the Number of Maximal Infeasible Vectors

In this section we prove Theorem 2. We first need some notations and definitions.

Let $\mathcal{C} = \{x \in \mathbb{Z}^n \mid 0 \leq x \leq c\}$ be a box and let $f : \mathcal{C} \to \{0,1\}$ be a discrete binary function. The function f is called *monotone* if $f(x) \geq f(y)$ whenever $x \geq y$ and $x, y \in \mathcal{C}$. We denote by $T(f)$ and $F(f)$ the sets of all true and all false vectors of f, i.e.,

$$T(f) = \{x \in \mathcal{C} | f(x) = 1\} = (\min[f])^+, \quad F(f) = \{x \in \mathcal{C} | f(x) = 0\} = (\max[f])^-,$$

where $\min[f]$ and $max[f]$ are the sets of all minimal true and all maximal false vectors of f, respectively.

Let $\sigma \in \mathbb{S}_n$ be a permutation of the coordinates and let x, y be two n-vectors. We say that y is a *left-shift* of x and write $y \succeq_\sigma x$ if the inequalities

$$\sum_{j=1}^{k} y_{\sigma_j} \geq \sum_{j=1}^{k} x_{\sigma_j}$$

hold for all $k = 1, \ldots, n$. A discrete binary function $f : \mathcal{C} \to \{0,1\}$ is called *2-monotonic with respect to* σ if $f(y) \geq f(x)$ whenever $y \succeq_\sigma x$ and $x, y \in \mathcal{C}$. Clearly, $y \geq x$ implies $y \succeq_\sigma x$ for any $\sigma \in \mathbb{S}_n$, so that any 2-monotonic function is monotone.

The function f will be called *regular* if it is 2-monotonic with respect to the identity permutation $\sigma = (1, 2, ..., n)$. Any 2-monotonic function can be transformed into a regular one by appropriately re-indexing its variables. To simplify notations, we shall state Lemma 1 below for regular functions, i.e., we fix $\sigma = (1, 2, ..., n)$ in this lemma.

For a given subset $\mathcal{A} \subseteq \mathcal{C}$ let us denote by $\mathcal{A}^*$ all the vectors which are left-shifts of some vectors of $\mathcal{A}$, i.e., $\mathcal{A}^* = \{y \in \mathcal{C} \mid y \succeq x \text{ for some } x \in \mathcal{A}\}$. Clearly, $T(f) = (\min[f])^*$ for a regular function f (in fact, the subfamily of *right-most* vectors of $\min[f]$ would be enough to use here.)

Given monotone discrete functions f and g, we call g a *regular majorant* of f, if $g(x) \geq f(x)$ for all $x \in \mathcal{C}$, and g is regular. Clearly, $T(g) \supseteq (\min[f])^*$ must hold in this case, and the discrete function h defined by $T(h) = (\min[f])^*$ is the unique minimal regular majorant of f.

For a vector $x \in \mathcal{C}$, and for an index $1 \leq k \leq n$, let the vectors $x^{(k]}$ and $x^{[k)}$ be defined by

$$x_j^{(k]} = \begin{cases} x_j & \text{for } j \leq k, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$x_j^{[k)} = \begin{cases} x_j & \text{for } j \geq k, \\ 0 & \text{otherwise.} \end{cases}$$

Let us denote by e the n-vector of all 1's, let e_j denote the j^{th} unit vector, $j = 1, ..., n$, and let $p(x)$ denote the number of positive components of the vector $x \in \mathcal{C}$.

Lemma 1. *Given a monotone discrete binary function $f : \mathcal{C} \to \{0,1\}$ such that $f \not\equiv 0$, and a regular majorant $g \geq f$, we have the inequality*

$$|F(g) \cap \max[f]| \leq \sum_{x \in \min[f]} p(x). \tag{5}$$

Proof. Let us denote by h the unique minimal regular majorant of f. Then we have $F(g) \cap \max[f] \subseteq F(h) \cap \max[f]$, and hence it is enough to show the statement for $g = h$, i.e. when $T(g) = (\min[f])^*$.

For a vector $y \in \mathcal{C} \setminus \{c\}$ let us denote by $l = l_y$ the index of the last component which is less than c_l, i.e., $l = \max\{j \mid y_j < c_j\} \in \{1, \ldots, n\}$. We claim that for every $y \in F(h) \cap \max[f]$ there exists an $x \in \min[f]$ such that

$$y = x^{(l-1]} + (x_l - 1)e_l + c^{[l+1)}, \tag{6}$$

where $l = l_y$. To see this claim, first observe that $y \neq c$ because $y \in F(f)$ and $f \not\equiv 0$. Second, for any j with $y_j < c_j$ we know that $y + e_j \in T(f)$, by the definition of a maximal false point. Hence there exists a minimal true-vector $x \in \min[f]$ such that $x \leq y + e_l$ for $l = l_y$. We must have $x^{(l-1]} = y^{(l-1]}$, since if $x_i < y_i$ for some $i < l$, then $y \geq x + e_i - e_l \succeq x$ would hold, i.e. $y \succeq x$ would follow, implying $y \in (\min[f])^*$ and yielding a contradiction with $y \in F(h) = \mathcal{C} \setminus (\min[f])^*$. Finally, the definition of $l = l_y$ implies that $y^{[l+1)} = c^{[l+1)}$. Hence, our claim and the equality (6) follow.

The above claim implies that

$$F(h) \cap \max[f] \subseteq \{x^{(l-1]} + (x_l - 1)e_l + c^{[l+1)} \mid x \in \min[f],\ x_l > 0\},$$

and hence (5) and thus the lemma follow. □

Lemma 2. *Let $f : \mathcal{C} \to \{0,1\}$ be a monotone discrete binary function such that $f \not\equiv 0$ and*

$$x \in T(f) \quad \Rightarrow \quad \alpha x \stackrel{\text{def}}{=} \alpha_1 x_1 + \ldots \alpha_n x_n \geq \beta, \tag{7}$$

where $\alpha = (\alpha_1, \ldots, \alpha_n)$ is a given real vector and β is a real threshold. Then

$$|\{x \in \mathcal{C} \mid \alpha x < \beta\} \cap \max[f]| \leq \sum_{x \in \min[f]} p(x).$$

Proof. Suppose that some of the weights $\alpha_1, \ldots, \alpha_n$ are negative, say $\alpha_1 < 0, \ldots, \alpha_k < 0$ and $\alpha^{[k+1)} \geq 0$. Since $\alpha x \geq \beta$ for any $x \in T(f)$ and since f is monotone, we have $x \in T(f) \Rightarrow \alpha^{[k+1)} x \geq \beta - \alpha^{(k]} c^{(k]}$. For any $x \in \mathcal{C}$ we also have $\{x \mid \alpha x < \beta\} \subseteq \{x \mid \alpha^{[k+1)} x < \beta - \alpha^{(k]} c^{(k]}\}$. Hence it suffices to prove the lemma for the non-negative weight vector $\alpha^{[k+1)}$ and the threshold $\beta - \alpha^{(k]} c^{(k]}$. In other words, we can assume without loss of generality that the original weight vector α is non-negative.

Let $\sigma \in \mathbb{S}^n$ be a permutation such that $\alpha_{\sigma_1} \geq \alpha_{\sigma_2} \geq \cdots \geq \alpha_{\sigma_n} \geq 0$. Then the threshold function

$$g(x) = \begin{cases} 1 & \text{if } \alpha x \geq \beta, \\ 0 & \text{otherwise.} \end{cases}$$

is 2-monotonic with respect to σ. By (7), we have $g \geq f$ for all $x \in \mathcal{C}$, i.e., g majorates f. In addition, $F(g) = \{x \in \mathcal{C} \mid \alpha x < \beta\}$, and hence Lemma 2 follows from Lemma 1. □

We are now ready to show inequality (4) and finish the proof of Theorem 2. Given a non-empty set $\mathcal{X} \subseteq \mathcal{F}_{A,b,c}$, consider the monotone discrete function $f : \mathcal{C} \to \{0, 1\}$ defined by the condition $\min[f] = \mathcal{X}$. Since (1) is monotone, any true vector of f also satisfies (1):

$$x \in T(f) \quad \Rightarrow a_{k1}x_1 + \ldots + a_{kn}x_n \geq b_k$$

for all $k = 1, \ldots, r$. In addition, $f \not\equiv 0$ because $\mathcal{X} \neq \emptyset$. Thus, by Lemma 2 we have the inequalities

$$|\{x \mid a_{k1}x_1 + \ldots + a_{kn}x_n < b_k\} \cap \max[f]| \leq \sum_{x \in \mathcal{X}} p(x) \tag{8}$$

for each $k = 1, ..., r$. Now, from $\max[f] = \mathcal{I}(\mathcal{X})$ we deduce that

$$\mathcal{I}(\mathcal{F}_{A,b,c}) \cap \mathcal{I}(\mathcal{X}) \subseteq \bigcup_{k=1}^{r} \{x \mid a_{k1}x_1 + \ldots + a_{kn}x_n < b_k\} \cap \max[f],$$

and thus (4) and the theorem follow by (8).

3 Generating Minimal Integer Solutions via Integral Dualization

The proof of Theorem 3 has two ingredients. First, we show that given a monotone system (1), the sets $\mathcal{I}(\mathcal{F}_{A,b,c})$ and $\mathcal{F}_{A,b,c}$ can be *jointly* enumerated by iteratively solving the dualization problem $DUAL(\mathcal{C}, \mathcal{A}, \mathcal{B})$ introduced in Theorem 3. Second, we invoke Theorem 2 and argue that since the number of maximal infeasible vectors is relatively small, the generation of $\mathcal{F}_{A,b,c}$ polynomially reduces to the joint generation of $\mathcal{I}(\mathcal{F}_{A,b,c})$ and $\mathcal{F}_{A,b,c}$.

3.1 Joint Generation of Dual Subsets in an Integral Box

Let $\mathcal{F} = \mathcal{F}_{A,b,c}$ be the set of minimal integral vectors for (1), and consider the following problem of jointly generating all points of $\mathcal{F}$ and $\mathcal{I}(\mathcal{F})$:

$GEN(\mathcal{F}, \mathcal{I}(\mathcal{F}), \mathcal{A}, \mathcal{B})$**:** *Given two explicitly listed collections* $\mathcal{A} \subseteq \mathcal{F}$ *and* $\mathcal{B} \subseteq \mathcal{I}(\mathcal{F})$*, either find a new point in* $(\mathcal{F} \setminus \mathcal{A}) \cup (\mathcal{I}(\mathcal{F}) \setminus \mathcal{B})$*, or prove that these collections are complete:* $(\mathcal{A}, \mathcal{B}) = (\mathcal{F}, \mathcal{I}(\mathcal{F}))$.

Proposition 2. *Problem* $GEN(\mathcal{F}, \mathcal{I}(\mathcal{F}), \mathcal{A}, \mathcal{B})$ *can be solved in time* $poly(n, |\mathcal{A}|, |\mathcal{B}|, \log \|c\|_\infty) + T_{dual}$, *where* T_{dual} *denotes the time required to solve problem* $DUAL(\mathcal{C}, \mathcal{A}, \mathcal{B})$.

Proof. The reduction is via the following Algorithm $\mathcal{J}$:

Step 1. Check whether $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$. If there is an $x \in \mathcal{B} \setminus \mathcal{I}(\mathcal{A})$, then $x \notin \mathcal{F}^+$ because $x \in \mathcal{B} \subseteq \mathcal{I}(\mathcal{F})$. This and the inclusion $\mathcal{A} \subseteq \mathcal{F}$ imply that $x \notin \mathcal{A}^+$. Since $x \notin \mathcal{I}(\mathcal{A})$, we can find a coordinate $j \in \{1, \ldots, n\}$ for which $y = x + e_j \notin \mathcal{A}^+$. By the maximality of x in $\mathcal{C} \setminus \mathcal{F}^+$, y belongs to $\mathcal{F}^+$ and therefore, there must exist a $z \in \mathcal{F}$ such that $z \leq y$. Since $z \notin \mathcal{A}^+$, we have $z \in \mathcal{F} \setminus \mathcal{A}$, i.e., z is a *new* minimal integral vector in $\mathcal{F}$ which can be found in $poly(n, |\mathcal{A}|, |\mathcal{B}|, \log \|c\|_\infty)$ time by performing coordinate binary searches on the box $\{z \in \mathbb{Z}^n \mid 0 \leq z \leq y\}$.

Step 2 is similar to the previous step: we check whether $\mathcal{A} \subseteq \mathcal{I}^{-1}(\mathcal{B})$, where $\mathcal{I}^{-1}(\mathcal{B})$ is the set of integral vectors minimal in $\mathcal{C} \setminus \mathcal{B}^-$. If $\mathcal{A}$ contains an element that is not minimal in $\mathcal{C} \setminus \mathcal{B}^-$, we can find a new point in $\mathcal{I}(\mathcal{F}) \setminus \mathcal{B}$ and halt.

Step 3. Suppose that $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$ and $\mathcal{A} \subseteq \mathcal{I}^{-1}(\mathcal{B})$. Then $(\mathcal{A}, \mathcal{B}) = (\mathcal{F}, \mathcal{I}(\mathcal{F})) \Leftrightarrow \mathcal{B} = \mathcal{I}(\mathcal{A})$. (To see this, assume that $\mathcal{B} = \mathcal{I}(\mathcal{A})$, and suppose on the contrary that there is an $x \in \mathcal{F} \setminus \mathcal{A}$. Since $x \notin \mathcal{A} = \mathcal{I}^{-1}(\mathcal{B})$ and $x \notin \mathcal{B}^- \subseteq \mathcal{I}(\mathcal{F})^-$, there must exist a $y \in \mathcal{I}^{-1}(\mathcal{B}) = \mathcal{A} \subseteq \mathcal{F}$ such that $y \leq x$. Hence we get two distinct elements $x, y \in \mathcal{F}$ such that $y \leq x$, which contradicts the definition of $\mathcal{F}$. The existence of an $x \in \mathcal{I}(\mathcal{F}) \setminus \mathcal{B}$ leads to a similar contradiction.) To check the condition $\mathcal{B} = \mathcal{I}(\mathcal{A})$, we solve problem $DUAL(\mathcal{C}, \mathcal{A}, \mathcal{B})$. If $\mathcal{B} \neq \mathcal{I}(\mathcal{A})$, we obtain a new point $x \in \mathcal{I}(\mathcal{A}) \setminus \mathcal{B}$. By (3), either $x \in \mathcal{F}^+$, or $x \in \mathcal{I}(\mathcal{F})^-$ and we can decide which of these two cases holds by checking the feasibility of x for (1). In the first case, we obtain a new point $y \in \{x\}^- \cap (\mathcal{F} \setminus \mathcal{A})$ by performing binary searches on the coordinates of the box $\{y \in \mathbb{Z}^n \mid 0 \leq y \leq x\}$. In the second case, a new point in $\{x\}^+ \cap (\mathcal{I}(\mathcal{F}) \setminus \mathcal{B})$ can be obtained by searching the box $\{y \in \mathbb{Z}^n \mid x \leq y \leq c\}$. □

Let $\mathcal{F} \subseteq \mathcal{C}$ be an arbitrary antichain, i.e., a system of integral vectors such that $x \not\leq y$ for any two distinct elements $x, y \in \mathcal{F}$. It is easy to see that Algorithm $\mathcal{J}$ and Proposition 2 can be used for any class of antichains $\mathcal{F}$ defined by a polynomial-time membership oracle for $\mathcal{F}^+$.

3.2 Uniformly Dual-Bounded Antichains

Extending the definition of dual-bounded hypergraphs in [5], we say that (a class of antichains) $\mathcal{F} \subseteq \mathcal{C}$ is *uniformly dual-bounded* if there exists a polynomial p such that, for any nonempty subset $\mathcal{X} \subseteq \mathcal{F}$, we have

$$|\mathcal{I}(\mathcal{F}) \cap \mathcal{I}(\mathcal{X})| \leq p(|\mathcal{X}|).$$

Proposition 3. *Suppose that* $\mathcal{F}$ *is uniformly dual-bounded and there exists a polynomial-time membership oracle for* $\mathcal{F}^+$. *Then problem* $GEN(\mathcal{F})$ *is polynomial-time reducible to problem* $DUAL(\mathcal{C}, \mathcal{A}, \mathcal{B})$.

Proof. Given a set $\mathcal{X}$ in $\mathcal{F}$, we repeatedly run Algorithm $\mathcal{J}$ until it either produces a new element in $\mathcal{F} \setminus \mathcal{X}$ or proves that $\mathcal{X} = \mathcal{F}$ by generating the entire family $\mathcal{I}(\mathcal{F})$. By Step 1, as long as Algorithm $\mathcal{J}$ outputs elements of $\mathcal{I}(\mathcal{F})$, these elements also belong to $\mathcal{I}(\mathcal{X})$, and hence the total number of such elements does not exceed $p(|\mathcal{X}|)$. □

By Theorem 2, the set of minimal integral solutions to any monotone system of linear inequalities is uniformly-dual bounded, and hence Theorem 3 is a corollary of Proposition 3.

4 Dualization in Products of Chains

Let $\mathcal{C} \stackrel{\text{def}}{=} \mathcal{C}_1 \times \ldots \times \mathcal{C}_n$ be an integer box defined by the product of n chains $\mathcal{C}_i = [l_i : u_i]$ where $l_i, u_i \in \mathbb{Z}$ are, respectively, the lower and upper bounds of chain $\mathcal{C}_i$. Given an antichain $\mathcal{A} \subseteq \mathcal{C}$, and an antichain $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$, we say that $\mathcal{B}$ is *dual to* $\mathcal{A}$ if $\mathcal{B} = \mathcal{I}(\mathcal{A})$, i.e., $\mathcal{B}$ contains all the maximal elements of $\mathcal{C} \setminus \mathcal{A}^+$. If $\mathcal{C}$ is the unit cube, we obtain the familiar notion of dual hypergraphs, where $\mathcal{I}(\mathcal{A})$ becomes the complementary set of the transversal hypergraph of $\mathcal{A}$. In this section, we show how to extend the hypergraph dualization algorithm of [8] to arbitrary systems $\mathcal{A}$ of integral vectors in a box $\mathcal{C}$.

As in [8], we shall analyze the running time of the algorithm in terms of the "volume" $v = v(\mathcal{A}, \mathcal{B}) \stackrel{\text{def}}{=} |\mathcal{A}||\mathcal{B}|$ of the input problem. In general, a given problem will be decomposed into a number of subproblems which we solve recursively. Since we have assumed that $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$, (3) implies that the following condition holds for the original problem and all subsequent subproblems:

$$a \not\leq b, \quad \text{for all } a \in \mathcal{A}, b \in \mathcal{B}. \tag{9}$$

Let $R(v) = R(v(\mathcal{A}, \mathcal{B}))$ denote the number of subproblems that have to be solved in order to solve the original problem, and let m denote $|\mathcal{A}| + |\mathcal{B}|$, and $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. We start with the following proposition that provides the base case for recursion.

Proposition 4. *Suppose* $\min\{|\mathcal{A}|, |\mathcal{B}|\} \leq const$, *then problem* $DUAL(\mathcal{C}, \mathcal{A}, \mathcal{B})$ *is solvable in polynomial time.*

Proof. Let us assume without loss of generality that $\mathcal{B} = \{b^1, \ldots, b^k\}$, for some constant k. For $t \in [n]^k$ and $i \in [n]$, let $I_i^t = \{j \in [k] \mid t_j = i\}$. Then $\mathcal{C} = \mathcal{A}^+ \cup \mathcal{B}^-$ *if and only if* for every $t \in [n]^k$ for which

$$b_i^j \neq u_i, \text{ for all } i \in [n], j \in I_i^t, \tag{10}$$

there exists an $a \in \mathcal{A}$ such that

$$a_i \leq \max\{b_i^j + 1 \mid j \in I_i^t\} \text{ if } I_i^t \neq \emptyset, \text{ and } a_i = l_i \text{ otherwise.} \tag{11}$$

To see this, assume first that $\mathcal{C} = \mathcal{A}^+ \cup \mathcal{B}^-$ and consider any $t \in [n]^k$ such that (10) holds. Let $x \in \mathcal{C}$ be defined by taking $x_i = \max\{b_i^j + 1 \mid j \in I_i^t\}$ if $I_i^t \neq \emptyset$,

and $x_i = l_i$ otherwise. Then $x \in \mathcal{C} \setminus \mathcal{B}^-$ and hence $x \in \mathcal{A}^+$, implying that there is an $a \in \mathcal{A}$ satisfying (11). On the other hand, let us assume that for every $t \in [n]^k$ satisfying (10), there is an $a \in \mathcal{A}$ for which (11) holds. Consider an $x \in \mathcal{C} \setminus \mathcal{B}^-$, then there must exist, for every $j \in [k]$, a $t_j \in [n]$, such that $x_{t_j} \geq b^j_{t_j} + 1$. Clearly $t = (t_1, \ldots, t_k) \in [n]^k$ satisfies (10), and therefore, there is an $a \in \mathcal{A}$ such that $a_i \leq \max\{b^j_i + 1 \mid j \in I^t_i\} \leq x_i$ if $I^t_i \neq \emptyset$, and $a_i = l_i$ otherwise. This gives $x \in \mathcal{A}^+$. □

Remark. Having found an $x \in \mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$, it is always possible to extend it to a maximal point with the same property in $O(nm \log m)$ time as follows. Let $\mathcal{Q}_i = \{a_i - 1 \mid a \in \mathcal{A}\} \cup \{x_i, u_i\}$, $i = 1, \ldots, n$, and assume that this list is kept in sorted order for each i. For $i = 1, \ldots, n$, we iterate $x_i \leftarrow \max\{z \in \mathcal{Q}_i \mid (x_1, \ldots, x_{i-1}, z, x_{i+1}, \ldots, x_n) \notin \mathcal{A}^+\}$. Then the resulting point x is maximal in $\mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$.

Now given two integral antichains $\mathcal{A}, \mathcal{B}$ that satisfy the necessary duality condition (9), we proceed as follows:

Step 1. If $\min\{|\mathcal{A}|, |\mathcal{B}|\} \leq 2$, the duality of $\mathcal{A}$ and $\mathcal{B}$ can be tested in $O(n^3 m)$ time using Proposition 4.

Step 2. For each $k \in [n]$:

1. if $a_k > u_k$ for some $a \in \mathcal{A}$ ($b_k < l_k$ for some $b \in \mathcal{B}$), then a (respectively, b) can be clearly discarded from further consideration;

2. if $a_k < l_k$ for some $a \in \mathcal{A}$ ($b_k > u_k$ for some $b \in \mathcal{B}$), we set $a_k \leftarrow l_k$ (respectively, $b_k \leftarrow u_k$). Note that the duality condition (9) continues to hold after such replacements.

Thus we may assume for next steps that $\mathcal{A}, \mathcal{B} \subseteq \mathcal{C}$.

Step 3. Let $a^o \in \mathcal{A}$, $b^o \in \mathcal{B}$. By (9), there exists an $i \in [n]$, such that $a^o_i > b^o_i$. Assume, with no loss of generality, that $i = 1$ and set $\mathcal{C}'_1 \leftarrow [a^o_1 : u_1]$, $\mathcal{C}''_1 \leftarrow [l_1 : a^o_1 - 1]$. (Alternatively, we may set $\mathcal{C}''_1 \leftarrow [l_1 : b^o_1]$ and $\mathcal{C}'_1 \leftarrow [b^o_1 + 1 : u_1]$.) Define

$$\mathcal{A}'' = \{a \in \mathcal{A} \mid a_1 < a^o_1\}, \qquad \mathcal{A}' = \mathcal{A} \setminus \mathcal{A}'', \qquad \epsilon^{\mathcal{A}}_1 = \frac{|\mathcal{A}'|}{|\mathcal{A}|},$$
$$\mathcal{B}' = \{b \in \mathcal{B} \mid b_1 \geq a^o_1\}, \qquad \mathcal{B}'' = \mathcal{B} \setminus \mathcal{B}', \qquad \epsilon^{\mathcal{B}}_1 = \frac{|\mathcal{B}''|}{|\mathcal{B}|}.$$

Observe that $\epsilon^{\mathcal{A}}_1 > 0$ and $\epsilon^{\mathcal{B}}_1 > 0$ since $a^o \in \mathcal{A}'$ and $b^o \in \mathcal{B}''$.

Denoting by $\mathcal{C}' = \mathcal{C}'_1 \times \mathcal{C}_2 \times \ldots \times \mathcal{C}_n$, and $\mathcal{C}'' = \mathcal{C}''_1 \times \mathcal{C}_2 \times \ldots \times \mathcal{C}_n$ the two half-boxes of $\mathcal{C}$ induced by the above partitioning, it is then easy to see that $\mathcal{A}$ and $\mathcal{B}$ are dual in $\mathcal{C}$ *if and only if*

$$\mathcal{A}, \mathcal{B}' \text{ are dual in } \mathcal{C}', \text{ and} \tag{12}$$
$$\mathcal{A}'', \mathcal{B} \text{ are dual in } \mathcal{C}''. \tag{13}$$

Step 4. Define $\epsilon(v) = 1/\chi(v)$, where $\chi(v)^{\chi(v)} = v = v(\mathcal{A}, \mathcal{B})$. If $\min\{\epsilon^{\mathcal{A}}_1, \epsilon^{\mathcal{B}}_1\} > \epsilon(v)$, we use the decomposition rule given above, which amounts to solving recursively two subproblems (12), (13) of respective volumes:

$$v(\mathcal{A}, \mathcal{B}') = |\mathcal{A}||\mathcal{B}'| = |\mathcal{A}|(1 - \epsilon^{\mathcal{B}}_1)|\mathcal{B}| = (1 - \epsilon^{\mathcal{B}}_1)v(\mathcal{A}, \mathcal{B}) \leq (1 - \epsilon(v))v,$$
$$v(\mathcal{A}'', \mathcal{B}) = |\mathcal{A}''||\mathcal{B}| = (1 - \epsilon^{\mathcal{A}}_1)|\mathcal{A}||\mathcal{B}| = (1 - \epsilon^{\mathcal{A}}_1)v(\mathcal{A}, \mathcal{B}) \leq (1 - \epsilon(v))v.$$

This gives rise to the recurrence

$$R(v) \leq 1 + R((1-\epsilon_1^{\mathcal{B}})v) + R((1-\epsilon_1^{\mathcal{A}})v) \leq 1 + 2R((1-\epsilon(v))v).$$

Step 5. Let us now suppose that $\epsilon_1^{\mathcal{B}} \leq \epsilon(v)$. In this case, we begin by solving subproblem (12). If $\mathcal{A}, \mathcal{B}'$ are not dual in $\mathcal{C}'$, we get a point x maximal in $\mathcal{C}' \setminus [\mathcal{A}^+ \cup (\mathcal{B}')^-]$, and we are done. Otherwise we claim that

$$\mathcal{A}'', \mathcal{B} \text{ are dual in } \mathcal{C}'' \iff \forall a \in \widetilde{\mathcal{A}} : \mathcal{A}'', \mathcal{B}'' \text{ are dual in } \mathcal{C}''(a), \tag{14}$$

where $\widetilde{\mathcal{A}} = \{a \in \mathcal{A} \mid a_1 \leq a_1^o\}$, and $\mathcal{C}''(a) = \mathcal{C}_1'' \times [a_2 : u_2] \times \ldots \times [a_n : u_n]$.

Proof of (14). The forward direction does not use (12). Suppose that there is an $x \in \mathcal{C}''(a) \setminus [(\mathcal{A}'')^+ \cup (\mathcal{B}'')^-]$ for some $a \in \widetilde{\mathcal{A}}$, then $x_i \geq a_i$, for $i = 2, \ldots, n$. If $x \in (\mathcal{B}')^-$, i.e., $x \leq b$ for some $b \in \mathcal{B}'$, then by the definition of $\mathcal{B}'$, $b_1 \geq a_1^o$. On the other hand, $a \in \widetilde{\mathcal{A}}$ implies that $a_1 \leq a_1^o$. But then,

$$(a_1, a_2, \ldots, a_n) \leq (a_1^o, x_2, \ldots, x_n) \leq (b_1, b_2, \ldots, b_n),$$

which contradicts the assumed duality condition (9). This shows that $x \in \mathcal{C}'' \setminus [(\mathcal{A}'')^+ \cup (\mathcal{B}' \cup \mathcal{B}'')^-]$.

For the other direction, let $x \in \mathcal{C}'' \setminus [(\mathcal{A}'')^+ \cup \mathcal{B}^-]$. Since $x \notin (\mathcal{B}')^-$ and $x = (x_1, x_2, \ldots, x_n) < y \overset{\text{def}}{=} (a_1^o, x_2, \ldots, x_n)$, the vector y is also not covered by $\mathcal{B}'$. Thus $y \in \mathcal{C}' \setminus (\mathcal{B}')^-$. We conclude therefore, assuming (12), that $y \in \mathcal{A}^+$, i.e., there is an $a \in \mathcal{A}$ such that $a \leq y$. But this implies that $a \in \widetilde{\mathcal{A}}$ and hence that $x \in \mathcal{C}''(a) \setminus [(\mathcal{A}'')^+ \cup (\mathcal{B}'')^-]$ for some $a \in \widetilde{\mathcal{A}}$. □

It follows by (14) that, once we discover that (12) holds, we can reduce the solution of subproblem (13) to solving $|\widetilde{\mathcal{A}}|$ subproblems, each of which has a volume of $v(|\mathcal{A}''|, |\mathcal{B}''|) \leq \epsilon_1^{\mathcal{B}} v(\mathcal{A}, \mathcal{B})$. Thus we obtain the recurrence

$$R(v) \leq 1 + R((1-\epsilon_1^{\mathcal{B}})v) + |\mathcal{A}| R(\epsilon_1^{\mathcal{B}} v) \leq R((1-\epsilon_1^{\mathcal{B}})v) + \frac{v}{2} R(\epsilon_1^{\mathcal{B}} v),$$

where the last inequality follows from $|\mathcal{A}| \leq v/3$ and $v \geq 9$.

Step 6. Finally, if $\epsilon_1^{\mathcal{A}} \leq \epsilon(v) < \epsilon_1^{\mathcal{B}}$, we solve subproblem (13), and if we discover that $\mathcal{A}'', \mathcal{B}$ are dual in $\mathcal{C}''$, we obtain the following rule, symmetric to (14):

$$\mathcal{A}, \mathcal{B}' \text{ are dual in } \mathcal{C}' \iff \forall b \in \widetilde{\mathcal{B}} : \mathcal{A}', \mathcal{B}' \text{ are dual in } \mathcal{C}'(b),$$

where $\widetilde{\mathcal{B}} = \{b \in \mathcal{B} \mid b_1 \geq a_1^o - 1\}$, and $\mathcal{C}'(b) = \mathcal{C}_1' \times [l_2 : b_2] \times \ldots \times [l_n : b_n]$. This reduces our original problem into one subproblem of volume $\leq (1-\epsilon_1^{\mathcal{A}})v$, plus $|\widetilde{\mathcal{B}}|$ subproblems, each of volume at most $\epsilon_1^{\mathcal{A}} v$, thus giving the recurrence

$$R(v) \leq 1 + R((1-\epsilon_1^{\mathcal{A}})v) + |\mathcal{B}| R(\epsilon_1^{\mathcal{A}} v) \leq R((1-\epsilon_1^{\mathcal{A}})v) + \frac{v}{2} R(\epsilon_1^{\mathcal{A}} v).$$

Using induction on $v \geq 9$, it can be shown that the above recurrences imply that $R(v) \leq v^{\chi(v)}$ (see [8]). As $\chi(m^2) < 2\chi(m)$ and $v(\mathcal{A}, \mathcal{B}) < m^2$, we get $\chi(v) < \chi(m^2) < 2\chi(m) \sim 2 \log m / \log \log m$. Let us also note that every step above can be implemented in at most $O(n^3 m)$ time, independent of the chains sizes $|\mathcal{C}_i|$. This establishes the bound stated in Theorem 4.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. I. Verkamo, Fast discovery of association rules, In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy eds., *Advances in Knowledge Discovery and Data Mining*, 307-328, AAAI Press, Menlo Park, California, 1996.
2. J. C. Bioch, Dualization, decision lists and identification of monotone discrete functions, Annals of Mathematics and Artificial Intelligence 24 (1998) 69-91.
3. J. C. Bioch and T. Ibaraki, Complexity of identification and dualization of positive Boolean functions, *Information and Computation* 123 (1995) 50-63.
4. E. Boros, K. Elbassioni, V. Gurvich and L. Khachiyan, An incremental RNC algorithm for generating all maximal independent sets in hypergraphs of bounded dimension. DIMACS Technical Report 2000-21, Rutgers University. To appear in *Parallel Processing Letters*.
5. E. Boros, V. Gurvich, L. Khachiyan and K.Makino, Generating partial and multiple transversals of a hypergraph, In: *Automata, Languanges and Programming, 27th International Colloquium, ICALP 2000* (Montanari, J.D.P. Rolim and E. Welzl, eds.), pp. 588-599 (Springer Verlag, Berlin, Heidelberg, New York, July 2000). An extended version of this paper is to appear in *SIAM J. Computing*.
6. Y. Crama, P. L. Hammer and T. Ibaraki, Cause-effect relationships and partially defined boolean functions, *Annals of Operations Research* 16 (1988) 299-326.
7. T. Eiter and G. Gottlob, Identifying the minimal transversals of a hypergraph and related problems, *SIAM Journal on Computing*, 24 (1995) 1278-1304.
8. M. L. Fredman and L. Khachiyan, On the complexity of dualization of monotone disjunctive normal forms, *J. Algorithms*, 21 (1996) 618-628.
9. D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen, Data mining, hypergraph transversals and machine learning. In: *Proceedings of the 16th ACM-SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, (1997) pp. 12-15.
10. D. S. Johnson, M. Yannakakis and C. H. Papadimitriou, On generating all maximal independent sets, *Information Processing Letters*, 27 (1988) 119-123.
11. E. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, Generating all maximal independent sets: NP-hardness and polynomial-time algorithms, *SIAM Journal on Computing*, 9 (1980) 558-565.
12. K. Makino and T. Ibaraki, Interor and exterior functions of Boolean functions, *Discrete Applied Mathematics*, 69 (1996) 209-231.
13. Mangasarian, Mathematical programming in machine learning, in G. Di. Pillo and F. Giannessi eds. Nonlinear Optimization and Applications (Plenum Publishing, New York, 1996) 283-295.
14. A. Prékopa, *Stochastic Programming*, (Kluwer, Dordrecht, 1995).

Division Is in Uniform TC^0

William Hesse*

Department of Computer Science
University of Massachusetts
Amherst, MA 01002
FAX: (001) 413 545-1249
whesse@cs.umass.edu

Abstract. Integer division has been known since 1986 [4,13,12] to be in slightly non-uniform TC^0, i.e., computable by polynomial-size, constant depth threshold circuits. This has been perhaps the outstanding natural problem known to be in a standard circuit complexity class, but not known to be in its uniform version. We show that indeed division is in uniform TC^0. A key step of our proof is the discovery of a first-order formula expressing exponentiation modulo any number of polynomial size.

1 Introduction

The exact complexity of integer division has been harder to pin down than the complexities of addition, subtraction, and multiplication. In 1986, Beame, Cook, and Hoover showed that iterated multiplication, and thus division, could be performed by Boolean circuits of logarithmic depth (NC^1 circuits) [4]. In 1987, Reif showed that these circuits could be implemented as constant depth circuits containing threshold gates (TC^0 circuits) [12,13]. Since then, the remaining issue has been the complexity of constructing these circuits. Division is the only prominent natural problem whose computation uses non-uniform circuits, circuits which require a non-trivial amount of computation for their construction.

The division problem discussed in this paper is the division of two n-bit integers, given in binary, yielding their integer quotient, also in binary. A related problem is the multiplication of n n-bit integers, computing their product as a binary integer. These problems are easily reduced to each other, so that a uniform circuit for one yields a uniform circuit for the other.

In this paper, we construct uniform constant depth circuits for division and iterated multiplication. We work within the framework of descriptive complexity, and show that there is a first-order formula using majority quantifiers that expresses division. This implies that there is an FO-uniform TC^0 circuit performing division [3]. First-order (FO) uniformity, equivalent to DLOGTIME uniformity, is the strongest uniformity requirement found to be generally applicable. A key step focuses on the one step of the TC^0 division computation not previously known to be expressible by a first order formula with majority quantifiers (an

* Supported by NSF grant CCR-9877078.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 104–114, 2001.

FO(M) formula). This is the problem of finding powers in the finite field $\mathbf{Z}_p$, the integers modulo a prime, where p has $O(\log n)$ bits. We show that there is a first-order formula without majority quantifiers computing powers in $\mathbf{Z}_p$. Thus this subproblem is in FO, and can be computed with uniform AC^0 circuits.

2 Definitions

We will express division as a predicate DIVISION(X, Y, i) which is true if and only if bit i of $\lfloor X/Y \rfloor$ is 1. We denote numbers with n or $n^{O(1)}$ bits by capital letters, and numbers with $O(\log n)$ bits by lowercase letters. We also refer to numbers with $O(\log n)$ bits as small, and those with $n^{O(1)}$ bits as large. We will always note the size of numbers with $(\log n)^{O(1)}$ bits explicitly. The iterated multiplication problem will be written as the predicate IMULT$(A_1, \ldots, A_n, i)$ which is true if bit i of $\prod_{j=1}^{n} A_j$ is 1; i ranges from 0 to n^2, and so has $2 \log n$ bits.

Though the size of the input to division is $2n + \log n$ and the input to iterated multiplication has size $n^2 + 2 \log n$, we will consider the input size, for all problems in this paper, to be n, as the circuit complexity classes and descriptive complexity classes we consider are closed under a polynomial change in the input size.

In this paper we produce simple logical formulas expressing these predicates. A problem is in the complexity class FO (first order) if the predicate corresponding to the decision problem can be expressed by a first order formula interpreted over a finite universe, the set of natural numbers $0, \ldots, n$. The inputs to the problem are encoded as relations over the universe, and are available to be used in the formula. The fixed numeric relations $<$ and BIT are also available[1]. For example, the n bits of the input X to DIVISION are represented by the values of a unary predicate $X()$ on the elements of the universe: $X(1), X(2), \ldots, X(n)$. An n^2 bit input can be represented by a binary predicate, so the inputs $A_1, \ldots, A_n$ to IMULT are represented as a binary predicate A. Short inputs to a problem, like i, the index of the result bit queried, may be represented by a constant in the range $0, \ldots, n$, which can also be regardedas a free variable. Since an FO or FO(M) formula over the universe $1, \ldots, n^k$ can be simulated by an equivalent formula over the universe $1, \ldots, n$, DIVISION and IMULT with inputs X, Y, and A_i having n^k bits, encoded by k-ary relations over $0, \ldots n$, are in the same descriptive complexity class as DIVISION and IMULT with n-bit inputs.

DIVISION and IMULT are provably not in FO, as parity is FO reducible to them, and parity is not in FO [8,9]. They will be shown to be in the class FO(M), problems described by first-order logic plus the majority quantifier. The majority quantifier (Mx) can appear anywhere that an $(\exists x)$ or a $(\forall x)$ can appear. The formula $(Mx)\varphi(x)$ is true iff $\varphi(j)$ is true for more than half the values $0 \leq j \leq n$. These quantifiers let us count the number of 1 bits in a string of length n; the counting quantifiers $(\exists! i\ x)$ are definable in terms of (Mx). These quantifiers are

[1] Following [10], we consider FO to include ordering and BIT. The BIT predicate allows us to look at the bits of numbers. BIT(i, x) is true if bit i of the number x written in binary is 1. This is equivalent to having addition and multiplication on numbers between 0 and n.

analogous to gates with n inputs that output 1 iff at least i of their inputs are 1, called threshold gates. We see next how an FO(M) formula is equivalent to a circuit containing threshold gates.

A TC^0 circuit is a constant-depth, polynomial-size circuit with AND, OR, NOT, and threshold gates with arbitrary fanin. If the type of each gate and the connections between gates can be computed by a deterministic logtime Turing machine, or equivalently by an FO formula, then the circuit is FO-uniform. The equivalence of FO-uniform TC^0 circuits and FO(M) formulas is shown by Barrington, Immerman, and Straubing in [3]. The Boolean function computed by an FO-uniform TC^0 circuit can be computed by an FO(M) formula, a first order formula using majority quantifiers, ordering, and BIT. The converse also holds; any FO(M) formula can be turned into a uniform TC^0 circuit. Here and throughout the paper, uniform will mean FO-uniform.

TC^0 is contained in the circuit complexity class NC^1, which contains all problems decided by Boolean circuits containing NOT gates, and AND and OR gates with two inputs, with $n^{O(1)}$ gates, n inputs, and depth $O(\log n)$. TC^0 contains the class AC^0 of constant depth polynomial size circuits without threshold gates. FO-uniform AC^0 circuits are equivalent to FO formulas with only existential and universal quantifiers, no majority quantifiers [3].

3 Previous Work

As stated in the introduction, Beame, Cook, and Hoover, et. al. gave NC^1 circuits deciding DIVISION and IMULT in 1986 [4]. They also gave a polynomial time algorithm for constructing the n'th circuit. Reif showed how to convert these to constant-depth threshold circuits a year later [13,12]. Immerman and Landau then observed that the construction was logspace uniform given the product of the first n^3 primes, implying that the full construction was TC^1 uniform [11].

These circuits were based on finding the remainders of the inputs on division by a set of small primes. The value of a number modulo a set of primes uniquely determines its value modulo the product of those primes. This is referred to as the Chinese remainder representation (CRR). The circuits work by converting the inputs to CRR, computing iterated products in that representation, and converting the output to binary. In the later 1990s, Chiu, Davida, and Litow devised new ways of computing in CRR that reduced the complexity of converting from CRR into binary [5,6]. These steps allowed them to construct logspace-uniform and NC^1-uniform TC^0 circuits for division and iterated multiplication.

Allender and Barrington reinterpreted those results in the framework of descriptive complexity, and showed that the only difficulty in expressing iterated multiplication and division in FO(M) was the difficulty of raising numbers to a power modulo a small prime [2]. The current paper completes this effort by showing that this power predicate lies in FO. As division is complete for FO(M) via FO Turing reductions, it is unlikely that the complexity of division can be further reduced.

4 Division Reduces to POW

The key problem examined by this paper is POW, the predicate expressing exponentiation modulo a prime. For a small prime p, and small arguments a, r, and b,

$$\mathrm{POW}(a, r, b, p) \iff a^r \equiv b \pmod{p} \ .$$

To be exact, we have a family of problems $\mathrm{POW}_{k \log n}(a, r, b, p)$ for $k = 1, 2, \ldots$, where the inputs to $\mathrm{POW}_{k \log n}$ have size $k \log n$. An input with $k \log n$ bits can be represented by a k-tuple of variables taking values in $0, \ldots, n$. Thus $\mathrm{POW}_{k \log n}$ is a $4k$-ary numeric relation. Though the inputs have $O(\log n)$ bits, we consider the descriptive complexity of this problem as if it had input size n. We ask whether this predicate can be represented by FO or FO(M) formulas over the universe $0, \ldots, n$.

Allender, Barrington, and the author showed that DIVISION and IMULT are in FO(M) if and only if POW is in FO(M) [2]. They did this by showing that DIVISION and IMULT are FO-Turing reducible to POW. A version of this proof, with additional simplifications, is in the full version of this paper. The predicate POW is used to convert inputs from binary to CRR, and to find discrete logarithms in the multiplicative group $\mathbf{Z}_p^*$ of integers mod p for primes p in the CRR basis.

FO-Turing reducibility in descriptive complexity classes is formally defined using generalized quantifiers in [10]. In the case of an FO-Turing reduction to POW, we shall not use this full formal definition, but a simpler characterization of FO-Turing reducibility to a relation. In the case of $\mathrm{POW}(a, r, b, p)$, which could be considered as a primitive numeric relation of arity $4k$ (if the inputs have $k \log n$ bits), we can express FO(M) Turing reducibility to POW by simply saying a predicate φ is FO(M) Turing reducible to POW if and only if there is an FO(M) formula with numeric relations $\leq$, BIT, and POW that expresses φ. This is equivalent to saying $\varphi \in \mathrm{FO}(\mathrm{M}, \mathrm{POW})$. Clearly, if POW is in FO(M), then FO(M,POW)= FO(M). We replace all uses of POW in a formula φ with the equivalent FO(M) formula. This is all we shall need to use about the reduction from DIVISION and IMULT to POW.

5 POW Is FO-Turing Reducible to $\mathrm{IMULT}_{O(\log n)}$ and $\mathrm{DIVISION}_{O((\log n)^2)}$

We now show that we can produce an FO formula deciding POW, provided that we allow the formula to use the results of certain smaller IMULT and DIVISION problems. These problems will have inputs constructed by FO formulas from the inputs to POW, or from the outputs of other small IMULT and DIVISION problems. This can be characterized as an FO-Turing reduction from POW to these smaller versions of IMULT and DIVISION. Later we will show that these smaller versions of IMULT and DIVISION are in FO(M), and then show that they are in FO.

The scaled versions of IMULT and DIVISION have $(\log n)^{O(1)}$-bit inputs. We still consider them as having input size n, however, so we shall define them as

$$\mathrm{IMULT}_{(\log n)^k}(A_1, \ldots, A_n, j) =$$

$$\mathrm{IMULT}(A_1, \ldots, A_n, j) \ \wedge \ (\forall i) A_i < 2^{(\log n)^k} \ \wedge \ (\forall i > (\log n)^k) A_i = 1$$

$$\mathrm{DIVISION}_{(\log n)^k}(X, Y, i) = \mathrm{DIVISION}(X, Y, i) \wedge X < 2^{(\log n)^k} \wedge Y < 2^{(\log n)^k} \ .$$

Thus we only have to give correct answers to the problems when the inputs are small. An FO Turing reduction to these problems is more complicated than an FO(M) Turing reduction to POW because the inputs to these problems have $\omega(\log n)$ bits and so must be given as relations, not as first-order variables. We shall only point out where these problems are used in our first-order expression for POW, and state the implications if we have FO or FO(M) expressions for them.

To show that POW is in FO, we will prove a more general lemma about finding powers in groups. This is interesting in its own right, and necessary for the extension to finding powers modulo prime power moduli. We consider a group to be given in FO if group elements are labeled by elements of the universe and the product operation is given by an FO formula. Note that the identity element and inverse operation can be defined in FO from the product operation. We can also continue to use arithmetic operations on the universe, considered as the numbers $0, \ldots, n$.

Lemma 1. *Finding small powers in any group of order n is FO Turing-reducible to finding the product of $\log n$ elements.*

Proof. Suppose we want to find a^r, where a is an element of a group of order n. We will compute a set of elements $a_1, \ldots, a_k$ and exponents $u, u_1, \ldots, u_k$ such that

$$a^r = a^u a_1^{u_1} \cdots a_k^{u_k}$$

and $u_i < 2 \log n$, $u < 2(\log n)^2$.

Step 1. We choose a set of $k = o(\log n)$ primes $d_1, \ldots, d_k$, such that $d_i < 2 \log n$ and d_i is relatively prime to n, for all i. We choose them such that $n < D = d_1 d_2 \cdots d_k < n^2$. We can do this with a first order formula by choosing the first $D > n$ such that D is square-free, D and n are relatively prime, and all prime factors of D are less than $2 \log n$. We can decide, given D, whether a number is one of our d_i or not. To compute the number k from D, and to find our list d_i as a relation between i and d_i, requires, for each prime $p_0 < 2 \log n$, counting the number of primes p dividing D which are less than p_0. We can do this using the BITSUM predicate, which counts the number of one bits in a $\log n$ bit number: $\mathrm{BITSUM}(x, y)$ is true if the binary representation of x contains y ones. This is shown to be in FO in [3].

Step 2. We calculate $a_i = a^{\lfloor n/d_i \rfloor}$ as follows:

First we calculate $n_i = n \bmod d_i$. Compute a^{-1} using the inverse operation. We find a^{-n_i} by multiplying n_i copies of a^{-1} together. This is one place where our Turing reduction to multiplication of $\log n$ group elements is used.

We can find $a^{\lfloor n/d_i \rfloor}$ by observing that

$$(a^{\lfloor n/d_i \rfloor})^{d_i} = a^{\lfloor n/d_i \rfloor d_i} = a^{n-(n \bmod d_i)} = a^{n-n_i} = a^{-n_i} \ .$$

Observe that there is exactly one group element x such that $x^{d_i} = a^{-n_i}$: Let d_i^{-1} be the multiplicative inverse to d_i mod n, i.e. that $d_i d_i^{-1} = mn + 1$ for some m. Then

$$x = x^{mn+1} = (x^{d_i})^{d_i^{-1}} = (a^{-n_i})^{d_i^{-1}} \ .$$

Thus we can find $a_i = a^{\lfloor n/d_i \rfloor}$ as the value of x in the expression

$$(\exists x)\ x^{d_i} = a^{-n_i}$$

We compute x^{d_i} using multiplication of $\log n$ elements. We could not compute $a^{\lfloor n/d_i \rfloor}$ directly as $(a^{-n_i})^{d_i^{-1}}$ since d_i^{-1} is not necessarily $O(\log n)$.

Step 3. Now we find the exponents $u, u_1, \ldots, u_k$ such that $a^u a_1^{u_1} \cdots a_k^{u_k} = a^r$.

Since $a_i = a^{\lfloor n/d_i \rfloor}$,

$$a_1^{u_1} \cdots a_k^{u_k} = a^{\left(\sum_{i=1}^k u_i \lfloor n/d_i \rfloor\right)} \ ,$$

$$\text{and since } a^r = a^u a_1^{u_1} \cdots a_k^{u_k} = a^{\left(u + \sum_{i=1}^k u_i \lfloor n/d_i \rfloor\right)} \ ,$$

$$u \equiv r - \sum_{i=1}^k u_i \lfloor \frac{n}{d_i} \rfloor \pmod{n} \ .$$

Thus, to make the final correction term a^u computable, we must make u as small as possible, and so we want to make $\sum_{i=1}^k u_i \lfloor n/d_i \rfloor \bmod n$ as close to r as possible. To approximate r as a linear combination of $\lfloor n/d_i \rfloor$, we use the Chinese remainder theorem.

Compute $f = \lfloor rD/n \rfloor$. This step requires r to have $O(\log n)$ bits. Using the Chinese remainder theorem, if we let $D_i = D/d_i$, and let $u_i = f D_i^{-1} \bmod d_i$, then

$$\sum_{i=1}^k u_i D_i \equiv f \pmod{D} \ . \text{ Let } m \text{ be s.t. } \sum_{i=1}^k u_i D_i = f + mD \ .$$

We can calculate u_i in FO, since we can guess the possibilities for D_i^{-1} in FO. Calculating u from the u_i involves a sum of k small numbers, which, since $k < \log n$, is in FO. This, again, uses the fact that BITSUM is in FO.

We now show that $u < (\log n)^2$. We calculate the difference between r and $\sum u_i \lfloor n/d_i \rfloor$:

$$\begin{aligned} \sum_{i=1}^k u_i \lfloor \frac{n}{d_i} \rfloor &= \sum_{i=1}^k \frac{u_i n}{d_i} - \sum_{i=1}^k (\frac{u_i n}{d_i} - u_i \lfloor \frac{n}{d_i} \rfloor) \\ &= \frac{n}{D} \sum_{i=1}^k u_i D_i - \sum_{i=1}^k u_i (\frac{n}{d_i} - \lfloor \frac{n}{d_i} \rfloor) \end{aligned}$$

$$= \frac{n}{D}(f + mD) - \sum_{i=1}^{k} u_i(\frac{n}{d_i} - \lfloor \frac{n}{d_i} \rfloor)$$

$$= \frac{n}{D}\lfloor \frac{rD}{n} \rfloor + nm - \sum_{i=1}^{k} u_i(\frac{n}{d_i} - \lfloor \frac{n}{d_i} \rfloor)$$

$$= r - \frac{n}{D}(\frac{rD}{n} - \lfloor \frac{rD}{n} \rfloor) + nm - \sum_{i=1}^{k} u_i(\frac{n}{d_i} - \lfloor \frac{n}{d_i} \rfloor) \ , \text{ so}$$

$$u = r - \sum_{i=1}^{k} u_i \lfloor \frac{n}{d_i} \rfloor \bmod n = \frac{n}{D}(\frac{rD}{n} - \lfloor \frac{rD}{n} \rfloor) + \sum_{i=1}^{k} u_i(\frac{n}{d_i} - \lfloor \frac{n}{d_i} \rfloor) \ .$$

The quantity $y - \lfloor y \rfloor$ is always between 0 and 1, and since $n/D < 1$, $u_i < 2 \log n$, and $k < \log n$, we see that $u < 2(\log n)^2 + 1$. Thus we can calculate a^u using two rounds of multiplying $\log n$ group elements.

Thus we have described group elements a_i and numbers u, u_i such that $a^u a_1^{u_1} \cdots a_k^{u_k} = a^r$ and the computation of $a^u a_1^{u_1} \cdots a_k^{u_k}$ is FO Turing reducible to the product of $\log n$ group elements. □

Because FO is closed under polynomial change in input size, and the product of $\log(n^k) = k \log n$ group elements is FO reducible to the product of $\log n$ group elements, we have

Corollary 1. *Finding powers in any group of order n^k is FO Turing-reducible to finding the product of $\log n$ elements.*

Representing a group of order n^k means representing elements as k-tuples of universe elements, and representing the product operation in FO.

We now apply this to the integers modulo p, where $p = O(n^k)$ is a prime. The multiplicative group $\mathbf{Z}_p^*$ contains the $p-1$ integers $1, \ldots, p-1$, and multiplication in this group is clearly first-order definable from multiplication and addition on $0, \ldots, n$. If a in $\mathrm{POW}(a, r, b, p)$ is zero, then we only need to check that b is zero. Otherwise, we find a^r in the multiplicative group Z_p^*. The product of $\log n$ group elements can be computed with $\mathrm{IMULT}_{k \log n}$ and $\mathrm{DIVISION}_{k \log^2 n}$, so we have the main lemma of this section:

Lemma 2. *POW is FO-Turing reducible to $IMULT_{O(\log n)}$ and $DIVISION_{O((\log n)^2)}$.*

6 $\mathrm{DIVISION}_{(\log n)^{O(1)}}$ and $\mathrm{IMULT}_{(\log n)^{O(1)}}$ Are in FO(M)

Since our end result is that DIVISION and IMULT are in FO(M), it should be no surprise that the logarithmically smaller versions $\mathrm{DIVISION}_{(\log n)^{O(1)}}$ and $\mathrm{IMULT}_{(\log n)^{O(1)}}$ are in FO(M). We will prove that these smaller versions are in FO(M) by reducing them to $\mathrm{POW}_{O(\log \log n)}$, and showing that $\mathrm{POW}_{O(\log \log n)}$ is in FO.

Just as we have introduced scaled versions of IMULT and DIVISION, we use a scaled version of POW:

$$\mathrm{POW}_{k \log\log n}(a, r, b, p) = \mathrm{POW}(a, r, b, p) \ \wedge \ a, r, b, p < 2^{k \log\log n}$$

The FO(M) Turing reduction of $\mathrm{IMULT}_{n^{O(1)}}$ to $\mathrm{POW} = \mathrm{POW}_{O(\log n)}$ shown by Allender et. al [2] scales to become an FO(M) Turing reduction of $\mathrm{IMULT}_{(\log n)^{O(1)}}$ to $\mathrm{POW}_{O(\log\log n)}$. This can be seen as follows: consider the FO(M) reduction on the problem with $(\log n)^{O(1)}$ input size, which is a Turing reduction using FO(M) formulas over the universe with $(\log n)^{O(1)}$ elements to the correspondingly scaled version of POW, $\mathrm{POW}_{O(\log\log n)}$. But any FO(M) formula over the smaller universe can be simulated by an FO(M) formula over the larger universe, so this is an FO(M) reduction from $\mathrm{IMULT}_{(\log n)^{O(1)}}$ to $\mathrm{POW}_{O(\log\log n)}$.

Showing that $\mathrm{POW}_{O(\log\log n)}$ is in FO can be done directly. Suppose the modulus p, the exponent r, and the base a all have fewer than $k \log\log n$ bits. The numbers $a_i = a^{\lfloor r/2^i \rfloor} \bmod p$, with i ranging from 0 to $k \log\log n$ can be guessed simultaneously, since there are $k \log\log n$ of them, each with $k \log\log n$ bits. An existential choice of a number x from 0 to $n-1$ can be thought of as a non-deterministic simultaneous guess of $\log n$ bits, so we can certainly simultaneously guess $(k \log\log n)^2$ bits. There is exactly one choice of the numbers $a_1, \ldots, a_{k \log\log n}$ such that the following conditions hold:

$$a_{k \log\log n} = a^{\lfloor r/2^{k \log\log n} \rfloor} = a^0 = 1 \text{ and } (\forall i)\ a_i \equiv a_{i+1}^2 a^{r_i} \pmod{p} \ ,$$

where r_i is bit i of r.

Extracting the numbers a_i out of our $\log n$ bit choice x and checking that they meet the above conditions can be done with an FO formula. Extracting a_0 gives us $a^r \bmod p$.

Thus we have concluded that $\mathrm{POW}_{O(\log\log n)}$ is in FO. Since we have an FO(M) Turing reduction from $\mathrm{IMULT}_{(\log n)^{O(1)}}$ and $\mathrm{DIVISION}_{(\log n)^{O(1)}}$ to $\mathrm{POW}_{O(\log\log n)}$, we can conclude

Theorem 1. *$\mathrm{IMULT}_{(\log n)^{O(1)}}$ and $\mathrm{DIVISION}_{(\log n)^{O(1)}}$ are in FO(M).*

7 DIVISION and IMULT Are in FO(M)

Since we have an FO Turing reduction from POW to $\mathrm{IMULT}_{O(\log n)}$ and $\mathrm{DIVISION}_{O(\log^2 n)}$, we can conclude that we have an FO(M) formula for POW. Finally, using the FO(M) Turing reduction from IMULT and DIVISION to POW, we arrive at our main result.

Theorem 2. *Iterated multiplication of n n-bit numbers and division of 2 n-bit numbers is in FO(M).*

By the equivalence of FO(M) to FO-uniform TC^0, we have

Corollary 2. *Iterated multiplication of n n-bit numbers and division of 2 n-bit numbers is in FO-uniform TC^0.*

As both of these classes are closed under polynomial change in the input size, these results also hold for inputs with $n^{O(1)}$ bits.

8 POW Is in FO

An additional result of the theorem that IMULT and DIVISION are in FO(M), is that $\text{IMULT}_{(\log n)^{O(1)}}$ and $\text{DIVISION}_{(\log n)^{O(1)}}$ are in FO. This is because any FO(M) formula over a universe $0, \ldots, \log n$ has an equivalent FO formula over the universe $0, \ldots, n$.

The fact that FO is closed under the introduction of counting quantifiers with polylogarithmic bounds is established in [1,7]. Since $\text{IMULT}_{(\log n)^{O(1)}}$ is equivalent to IMULT with input size $(\log n)^{O(1)}$, it is expressed by an FO(M) formula over $0, \ldots, \log n$. Therefore, $\text{IMULT}_{(\log n)^{O(1)}}$ is expressed by an FO formula, and similarly $\text{DIVISION}_{(\log n)^{O(1)}}$ is in FO, and we have

Theorem 3. *$\text{IMULT}_{(\log n)^{O(1)}}$ and $\text{DIVISION}_{(\log n)^{O(1)}}$ are in FO.*

This theorem gives us a tight bound on the size of cases of IMULT that are in FO. Since we know that $\text{PARITY}_{f(n)}$ is in FO iff $f(n) = (\log n)^{O(1)}$, from Håstad [9], and PARITY is easily FO many-one reducible to multiplication of two numbers, which is FO many-one reducible to IMULT of the same size, we can conclude that $\text{IMULT}_{f(n)}$ is in FO iff $f(n) = (\log n)^{O(1)}$.

Since our proof that POW was in FO(M) included an FO Turing reduction from POW to $\text{DIVISION}_{O((\log n)^2)}$ and $\text{IMULT}_{O(\log n)}$, and we now have FO formulas expressing $\text{DIVISION}_{O((\log n)^2)}$ and $\text{IMULT}_{O(\log n)}$, we can now conclude that POW is in FO. Since the restriction that the inputs to POW have $O(\log n)$ bits is equivalent to requiring that the inputs be in the range $0, \ldots, n$, we have our second main result.

Theorem 4. *The predicate POW(a, r, b, p) which is true iff $a^r \equiv b \pmod p$, with p prime, can be expressed by an FO formula over the universe $0, \ldots, n$, if $a, r, b, p \leq n$.*

This result can be extended to exponentiation modulo any small number n, not just modulo a prime. We can see that the equation

$$a^r \equiv b \pmod n$$

is true if and only if it is true modulo all the prime power factors of n:

$$a^r \equiv b \pmod{p^i}\ \forall p^i | n \ .$$

We can show that for a relatively prime to p^i, a is in the group $\mathbf{Z}^*_{p^i}$, and the above proof can be applied. If p divides a, then if $r > \log n$, $a^r \equiv 0 \pmod{p^i}$. If $r \leq \log n$, then $\text{IMULT}_{(\log n)^{O(1)}}$ can be applied. Since the prime power factors of a small number n can be found in FO, we have

Corollary 3. *The predicate $a^r \equiv b \pmod n$, with the inputs written in unary, is in FO.*

Finally, note that the property that any predicate expressible in FO over the universe $0, \ldots, n^k$ is expressible in FO over $0, \ldots, n$ lets us conclude that the predicate $a^r \equiv b (\bmod n)$ is in FO if the inputs have $O(\log n)$ bits, but not that it is in FO with inputs of $(\log n)^{O(1)}$ bits. This is different from the results we have for $\text{IMULT}_{(\log n)^{O(1)}}$ and $\text{DIVISION}_{(\log n)^{O(1)}}$.

9 Conclusions

Our main theorem states that division and iterated multiplication are in fully uniform TC^0. This is significant on its own and also because it eliminates the most important example of a problem known to be in a circuit complexity class, but not known to be in the corresponding uniform complexity class.

We also proved that exponentiation modulo a number is in FO when the inputs have $O(\log n)$ bits. This result was quite unexpected, since the problem was previously not even known to be in FO(M). It remains unknown if exponentiation modulo a number with $(\log n)^{O(1)}$ bits is in FO, or even in FO(M).

Finally, we have found a tight bound on the size of division and iterated multiplication problems that are in FO. We now know that these problems are in FO if and only if their inputs have $(\log n)^{O(1)}$ bits. Instances of the problems with larger inputs are known not to be in FO.

Acknowledgments. These results were found while working on [2] with Eric Allender and David Mix Barrington, who generously urged me to publish them separately.

References

1. M. Ajtai and M. Ben-Or. A theorem on probabilistic constant depth computations. In *ACM Symposium on Theory of Computing (STOC '84)*, pages 471–474, 1984. ACM Press.
2. E. Allender, D. A. Mix Barrington, and W. Hesse. Uniform circuits for division: Consequences and problems. To appear in *Proceedings of the 16th Annual IEEE Conference on Computational Complexity (CCC-2001)*, 2001. IEEE Computer Society.
3. D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990.
4. P. W. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 15(4):994–1003, 1986.
5. A. Chiu, G. Davida, and B. Litow. NC^1 division. online at http://www.cs.jcu.edu.au/∼bruce/papers/crr00_3.ps.gz.
6. G. I. Davida and B. Litow. Fast Parallel Arithmetic via Modular Representation. *SIAM Journal of Computing*, 20(4):756–765, 1991.
7. R. Fagin, M. M. Klawe, N. J. Pippenger, and L. Stockmeyer. Bounded-depth, polynomial-size circuits for symmetric functions. *Theoretical Computer Science*, 36(2-3):239–250, 1985.
8. M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. In *22nd Annual Symposium on Foundations of Computer Science*, 260–270, 1981. IEEE.
9. J. Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, 6–20, 1986.
10. N. Immerman. *Descriptive Complexity.* Springer-Verlag, New York, 1999.
11. N. Immerman and S. Landau. The complexity of iterated multiplication. *Information and Computation*, 116(1):103–116, 1995.

12. J. H. Reif. On threshold circuits and polynomial computation. In *Proceedings, Structure in Complexity Theory, Second Annual Conference*, pages 118–123, IEEE Computer Society Press.
13. J. H. Reif and S. R. Tate. On threshold circuits and polynomial computation. *SIAM Journal on Computing*, 21(5):896–908, 1992.

A Framework for Index Bulk Loading and Dynamization

Pankaj K. Agarwal*, Lars Arge**,
Octavian Procopiuc***, and Jeffrey Scott Vitter†

Center for Geometric Computing, Dept. of Computer Science,
Duke University, Durham, NC 27708-0129, USA.
{pankaj,large,tavi,jsv}@cs.duke.edu

Abstract. In this paper we investigate automated methods for externalizing internal memory data structures. We consider a class of balanced trees that we call *weight-balanced partitioning trees* (or *wp-trees*) for indexing a set of points in $\mathbb{R}^d$. Well-known examples of wp-trees include kd-trees, BBD-trees, pseudo-quad-trees, and BAR-trees. Given an efficient external wp-tree construction algorithm, we present a general framework for automatically obtaining a dynamic external data structure. Using this framework together with a new general construction (bulk loading) technique of independent interest, we obtain data structures with guaranteed good update performance in terms of I/O transfers. Our approach gives considerably improved construction and update I/O bounds for e.g. external kd-trees and BBD-trees.

1 Introduction

Both in the database and algorithm communities, much attention has recently been given to the development of I/O-efficient external data structures for indexing point data. A large number of external structures have been developed, reflecting the many different requirements put on such structures; small size, efficient query and update bounds, capability of answering a wide range of queries (mainly range and proximity queries), and simplicity. See recent surveys [2,14,

* Supported by Army Research Office MURI grant DAAH04–96–1–0013, by a Sloan fellowship, by NSF grants ITR–333–1050, EIA–9870724 and CCR–9732787 and by a grant from the U.S.-Israeli Binational Science Foundation.

** Supported in part by the National Science Foundation through ESS grant EIA–9870734, RI grant EIA–9972879 and CAREER grant EIA–9984099. Part of this work was done while visiting BRICS, University of Aarhus, Denmark.

*** Supported by the National Science Foundation through research grant EIA–9870734 and by the Army Research Office through MURI grant DAAH04–96–1–0013. Part of this work was done while visiting BRICS, University of Aarhus, Denmark.

† Supported in part by the National Science Foundation through research grants CCR–9877133 and EIA–9870734 and by the Army Research Office through MURI grant DAAH04–96–1–0013. Part of this work was done while visiting BRICS, University of Aarhus, Denmark.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 115–127, 2001.

27]. The proposed data structures can roughly be divided into two classes, namely simple and practical (often heuristics based) structures, for which worst-case query performance guarantees can only be given in the *static* case (if at all), and theoretically optimal but usually complicated *dynamic* structures. The first class of structures are often external versions of well-known simple internal memory structures.

In this paper, we develop a general mechanism for obtaining efficient external data structures from a general class of simple internal memory structures, such that the external structures are efficient in the *dynamic* case. Part of our result is a new general index construction (bulk loading) technique which is of independent interest.

1.1 Computational Model and Previous Results

In this paper we analyze data structures in the standard two-level external memory model defined by the following parameters [1,18]: N, the number of input elements, M, the number of elements that fit in main memory, and B, the number of elements that fit in one disk block, where $N \gg M$ and $1 \leq B \leq M/2$. One *I/O operation* (or simply *I/O*) in this model consists of reading one block of contiguous elements from disk into main memory or writing one block from main memory to disk. The measure of performance of an algorithm or data structure is the number of I/O operations it performs and the maximum disk space (blocks) it uses. For notational simplicity, we use $n = N/B$ and $m = M/B$ to denote the input size and memory size in units of data blocks.

Aggarwal and Vitter [1] developed algorithms for sorting a set of N elements in external memory in optimal $\Theta(n \log_m n)$ I/Os. Subsequently, I/O-efficient algorithms have been developed for large number of problems. Recently, many provably efficient (and often optimal) external data structures have also been developed. Ideally, an external data structure should use linear space, $O(n)$ blocks, and answer a query in $O(\log_B N+K/B)$ I/Os, where K is the number of elements reported by the query. These bounds are obtained by the B-tree data structure for one-dimensional range searching [8,11]. For two-dimensional range searching, $O(\sqrt{n} + K/B)$ is the best obtainable query bound with linear space [26]. Structures that use more than linear space are often infeasible in practical applications. Below we discuss the known external memory data structures most relevant to this paper. See [27,2] for complete surveys of known results.

One main challenge in the design of external data structures is obtaining good query performance in a dynamic environment. Early structures, such as the grid file [20], the various quad-trees [21,23], and the kdB-tree [22], were poorly equipped to handle updates. Later structures tried to employ various (heuristic) techniques to preserve the query performance and space usage under updates. They include the LSD-tree [16], the buddy tree [24], the hB-tree [19], and R-tree variants (see [14] and the references therein). These data structures are often the methods of choice in practical applications because they use linear space and reportedly perform well in practice. However, in a dynamic environment, the query time is high in the worst-case. The hB-tree (or holey brick tree),

for example, is based on the statically query-efficient kdB-tree, which combines the spatial query capabilities of the kd-tree [9] with the I/O-efficiency of the B-tree. While nodes in a kdB-tree represent rectangular regions of the space, nodes in an hB-tree represent so-called "holey bricks," or rectangles from which smaller rectangles have been cut out. This allows for the underlying B-tree to be maintained during updates (insertions). Unfortunately, a similar claim cannot be made about the underlying kd-tree, and thus good query-efficiency cannot be maintained.

Recently, a number of theoretical worst-case efficient dynamic external data structures have been developed. The cross-tree [15] and the O-tree [17], for example, both use linear-space, answer range queries in the optimal number of I/Os, and they can be updated I/O-efficiently. However, their practical efficiency has not been investigated, probably because a theoretical analysis shows that their average query performance is close to the worst-case performance. By contrast, the average-case performance of the kd-tree (and the structures based on it) is much better than the worst-case performance [25]. Other linear-space and query and update optimal external data structures have been designed for special types of range queries, like 2- or 3-sided two-dimensional range queries and halfspace range queries (see e.g. [2,4]). The practical efficiency of these structures still has to be established.

In the database literature, the term bulk loading is often used to refer to the process of constructing an external data structure. Since bulk loading an index using repeated insertion is often highly non-efficient [3], the development of specialized bulk loading algorithms has received a lot of attention recently. Most work on bulk loading has concentrated on the R-tree (see [3,12] and the references therein).

1.2 Our Results

In Section 2 of this paper, we define a class of linear-space trees for indexing a set of points in $\mathbb{R}^d$. These so-called *wp-trees* generalize known internal memory data structures like kd-trees [9], pseudo-quad-trees [21], BBD-trees [7], and BAR trees [13]. We also show how a wp-tree can be efficiently mapped to external memory, that is, how it can be stored in external memory using $O(n)$ blocks such that a root-to-leaf path can be traversed I/O-efficiently. In Section 3, we then design a general technique for bulk loading wp-trees. Using this technique we obtain the first I/O-optimal bulk loading algorithms for kd-trees, pseudo-quad-trees, BBD-trees and BAR-trees. Our algorithms use $O(n \log_m n)$ I/Os while previously known algorithms use at least $\Omega(n \log_2 n)$ I/Os. Finally, in Section 4, we describe several techniques for making a wp-tree dynamic. Our techniques are based on dynamization methods developed for internal memory (*partial rebuilding* and the *logarithmic method*) but adapted for external memory. Together with our external bulk loading technique, they allow us to obtain provably I/O-efficient dynamic versions of structures like the kd-trees, pseudo-quad-trees, BBD-trees, and BAR-trees. Previously, no such structures were known.

2 The wp-Tree Framework

In this section, we introduce wp-trees and show how they can mapped to external memory. To simplify the presentation, we discuss our results in $\mathbb{R}^2$. They can all easily be generalized to higher dimensions.

Definition 1 A (β, δ, κ) *weight-balanced partitioning tree* (or *wp-tree*) on a set S of N points in $\mathbb{R}^2$ satisfies the following constraints:

1. Each node v corresponds to a region r_v in $\mathbb{R}^2$, called the extent of v. The extent of the root node is $\mathbb{R}^2$;
2. Each non-leaf node v has $\beta \geq 2$ children corresponding to a partition of r_v into β disjoint regions;
3. Each leaf node v stores exactly one point p from S inside r_v;
4. Let $w(v)$ be the *weight* of node v, defined as the number of data points stored in the subtree rooted at v, and let $v^{(\kappa)}$ be the κ'th ancestor of v. Then $w(v) \leq \delta w(v^{(\kappa)})$ for all nodes v and $v^{(\kappa)}$.

The wp-tree generalizes a number of internal memory data structures used to index point data sets: kd-trees [9], pseudo-quad-trees [21], BBD-trees [7], and BAR-trees [13] are all wp-trees.

Condition 4 (the *weight condition*) insures that wp-trees are balanced; only a constant number of partition steps (κ) is required to obtain regions containing a fraction (δ) of the points.

Lemma 1. *The height of a wp-tree is at most* $\kappa(\log_{1/\delta} N + 1) - 1$.

We want to store a wp-tree on disk using $O(n)$ disk blocks so that a root-to-leaf path can be traversed I/O-efficiently. Starting with the root v we fill disk blocks with the subtree obtained by performing a breadth-first search traversal from v until we have traversed at most B nodes. We recursively block the tree starting in the leaves of this subtree. The *blocked wp-tree* obtained in this way can be viewed as a fanout $\Theta(B)$ tree with each disk block corresponding to a node. We call these nodes *block nodes* in order to distinguish them from wp-tree nodes. The leaf block nodes of the blocked wp-tree are potentially underfull (contain less than $\Theta(B)$ wp-tree nodes), and thus $O(N)$ blocks are needed to block the tree in the worst case. To alleviate this problem, we let certain block nodes share the same disk block. More precisely, if v is a non-leaf block node, we reorganize all v's children that are leaf block nodes, such that at most one disk block is non-full. This way we only use $O(n)$ disk blocks. Since each non-leaf block node contains a subtree of height $O(\log_2 B)$ we obtain the following.

Lemma 2. *A blocked wp-tree* $\mathcal{T}$ *is a multi-way tree of height* $O(\log_B N)$. *It can be stored using* $O(n)$ *blocks.*

2.1 The Restricted wp-Tree

The wp-tree definition emphasizes the structure of the tree more than the geometry of the partitioning. The dynamization methods we will discuss in Section 4

can be applied to any wp-tree. However, without specifying the properties of the partitioning, we cannot quantify the update and query I/O-bounds obtained using these methods. Therefore we now restrict the definition of a wp-tree by adding geometric constraints on the extent of a node and the partitioning methods used. On the one hand the resulting class of *restricted wp-trees* is general enough to encompass many interesting data structures, and on the other hand it is restrictive enough to allow us to prove general bulk loading, update, and query bounds.

Definition 2 A *restricted* (β, δ, κ) *wp-tree* is a (β, δ, κ) wp-tree in which each node v satisfies the following constraints:

1. The extent r_v of v is the region lying between two convex polygons; $r_v = b_O \setminus b_I$. The *inner polygon* b_I must be completely inside the *outer polygon* b_O, and the orientations of edges forming b_I and b_O must be taken from a constant set of directions D.
2. The extents of the β children of v are obtained from r_v by applying the following *cut rules* a constant number of times:
   a) A *geometric cut* (ℓ). A geometric cut is a line ℓ along a direction $e \in D$ not intersecting b_I.
   b) A *rank cut* (e, α). A rank cut is a line ℓ along a direction $e \in D$. Let ℓ' be the line along e such that $\alpha w(v)$ of the $w(v)$ points corresponding to v are to the left of ℓ'. Then ℓ is the closest line to ℓ' not intersecting the interior of b_I.
   c) A *rectangle cut*. A rectangle cut can be applied to v only if b_I and b_O are both fat rectangles (i.e., the aspect ratio is at most 3) and $2b_I \subset b_O$. A rectangle cut is a fat rectangle b' such that $b_I \subset b' \subset b_O$ and both $b' \setminus b_I$ and $b_O \setminus b'$ contain at most $2w(v)/3$ points.

2.2 Examples of Restricted wp-Trees

Like wp-trees, restricted wp-trees generalize internal memory data structures like kd-trees, BBD-trees, pseudo-quad-trees, and BAR-trees. Below we further discuss kd-trees and BBD-trees. In the full paper we show how pseudo-quad-trees and BAR-trees are also captured by the restricted wp-tree definition.

The kd-tree. Introduced by Bentley [9], the kd-tree is a classical structure for answering range (or window) queries. It is a binary tree that represents a recursive decomposition of the space by means of hyperplanes orthogonal to the coordinate axes. In $\mathbb{R}^2$ the partition is by axes-orthogonal lines. Each partition line divides the point-set into two equal sized subsets. On even levels of the tree the line is orthogonal to the x-axis, while on odd levels it is orthogonal to the y-axis. These partitions are *rank cuts* $(e, 1/2)$, where e is orthogonal to the x- or y-axis. Thus the kd-tree is a restricted $(2, 1/2, 1)$ wp-tree.

The BBD-tree. The *balanced box decomposition tree*, or *BBD-tree*, was introduced by Arya et al [7] for answering approximate nearest-neighbor queries. Like the kd-tree, the BBD-tree is a binary tree representing a recursive decomposition

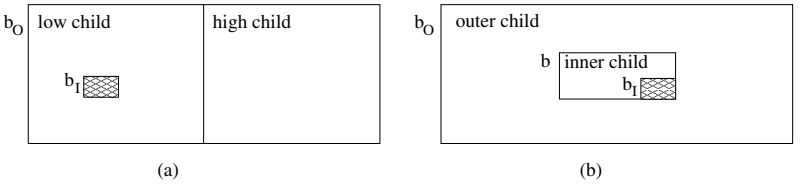


Fig. 1. BBD-tree partitions. (a) Split node. (b) Shrink node.

of the space. The region associated with a BBD-tree node is the set theoretic difference of two fat rectangles, b_I and b_O (with b_I included in b_O). More precisely, a BBD-tree consists of two types of nodes: *split nodes* and *shrink nodes*. In a *split node*, the partition is done using an axis-orthogonal line that cuts the longest side of b_O so that the resulting rectangles are fat and b_I lies entirely inside one of them—refer to Figure 1(a). In a *shrink node* v, the partition is done using a box rather than a line. This box b lies inside b_O, contains b_I, and determines the extent of the two children: $b \setminus b_I$ is the extent of the inner child and $b_O \setminus b$ is the extent of the outer child—refer to Figure 1(b). While split nodes reduce the geometric size, the box b used in shrink nodes is chosen so as to reduce the number of points by a factor of 1.5. By alternating split nodes and shrink nodes, both the geometric size and the number of points associated with each node decrease by a constant factor as we descend a constant number of levels in the BBD-tree (see [7] for details). It is easy to see that the split node uses a geometric cut and the shrink node uses a rectangle cut. In the full paper we show that a BBD-tree is a restricted (2,2/3,3) wp-tree.

3 Bulk Loading Restricted wp-Trees

In this section we describe an optimal algorithm for bulk loading (constructing) a blocked restricted wp-tree.

It is natural to bulk load a wp-tree using a top-down approach. For example, to construct a kd-tree on N points in $\mathbb{R}^2$ we first find the point with the median x-coordinate in $O(n)$ I/Os [1]. We then distribute the points into two sets based on this point and proceed recursively in each set, alternating between using the median x-coordinate and y-coordinate to define the distribution. This way each level of the wp-tree is constructed in a linear number of I/Os, so in total we use $O(n \log_2 n)$ I/Os to bulk load the tree. This is a factor of $\log_2 m$ larger than the optimal $O(n \log_m n)$ bound (the sorting bound). Intuitively, we need to construct $\Theta(\log_2 m)$ levels of the wp-tree in a linear number of I/Os—instead of just one—in order to obtain this bound. Doing so seems difficult because of the way the points are alternately split by x- and y-coordinates. Nevertheless, below we show how to bulk load a blocked restricted wp-tree, and thus a kd-tree, in $O(n \log_m n)$ I/Os.

To simplify the presentation, we present our restricted wp-tree bulk loading algorithm only for the case where $\beta = 2$ and where D contains the two directions orthogonal to the coordinate axes. The details of the general algorithm will be given in the full paper. Let S be a set of N points in $\mathbb{R}^2$. The first step in constructing a blocked wp-tree for S is to sort the N points twice: once according to their x-coordinate, and once according to their y-coordinate. Call the resulting ordered sets S_x and S_y, respectively. Next a recursive procedure **Bulk_load** is called on S_x and S_y. **Bulk_load** builds a subtree of height $\Theta(\log_2 m)$ in each recursive call, The main idea in the algorithm is to impose a grid on the set of input points and count the number of points in each grid cell. The grid counts allow us to compute partitions without reading all the points. More precisely, **Bulk_load** starts by dividing the current region (initially $\mathbb{R}^2$) into $t = \Theta(\min\{m, \sqrt{M}\})$ vertical slabs and t horizontal slabs, each containing N/t points—refer to Figure 2(a). These slabs form a $t \times t$ grid. Note that the grid size t^2 is at most M, and thus the grid can fit into internal memory. The number of points in each grid cell is then computed and stored in a matrix A in main memory. All three types of cuts can now be computed efficiently using A. A rank cut (e, α) for a node v, for example, is computed by first finding the slab E_k along e containing the cutting line. This can be done without performing I/Os. The exact cutting line can then be computed in $O(N/(Bt))$ I/Os by scanning the points in E_k. After a subtree $\mathcal{T}$ of height $\Theta(\log_2 t)$ is built, S_x and S_y are distributed into t sets each, corresponding to the leaves of $\mathcal{T}$ and the rest of the tree is built by calling **Bulk_load** recursively. The detailed **Bulk_load** procedure is given below.

procedure Bulk_load(S_x, S_y, v)

1. Divide S_x into t sets, corresponding to t vertical slabs $X_1, \ldots, X_t$, each containing $|S_x|/t$ points. Store the $t+1$ boundary x-coordinates in memory.
2. Divide S_y into t sets, corresponding to t horizontal slabs $Y_1, \ldots, Y_t$, each containing $|S_y|/t$ points. Store the $t+1$ boundary y-coordinates in memory.
3. The vertical and horizontal slabs form a grid. Let $C_{i,j}$ be the set of points in the grid cell formed at the intersection of the ith horizontal slab and the jth vertical slab. Create a $t \times t$ matrix A in memory. Scan S_x and compute the grid cell counts: $A_{i,j} = |\mathcal{C}_{i,j}|$, $1 \le i, j \le t$.
4. *Let* u=v.
5. a) If u is partitioned using a *geometric cut* orthogonal to the x-axis, determine the slab X_k containing the cut line ℓ using the boundary x-coordinates. Next scan X_k and, for each cell $\mathcal{C}_{j,k}$, $1 \le j \le t$, compute the counts of "subcells" $\mathcal{C}^{<}_{j,k}$ and $\mathcal{C}^{>}_{j,k}$ obtained by splitting cell $\mathcal{C}_{j,k}$ at ℓ—refer to Figure 2(b). Store these counts in main memory by splitting the matrix A into $A^{<}$ and $A^{>}$, containing the first k columns and the last $(t-k+1)$ columns of A, respectively (column k from matrix A appears in both $A^{<}$ and $A^{>}$). Then let $A^{<}_{j,k} = |\mathcal{C}^{<}_{j,k}|$ and $A^{>}_{j,1} = |\mathcal{C}^{>}_{j,k}|$, $1 \le j \le k$. Go to 5.(d).
   b) If u is partitioned using a *rank cut* orthogonal to the x-axis, first determine the slab X_k containing the cut line ℓ using A, then scan X_k to determine the exact position of the cut line. Next split A into $A^{<}$ and $A^{>}$ as above. A cut orthogonal to the y-axis is handled similarly. Go to 5.(d).

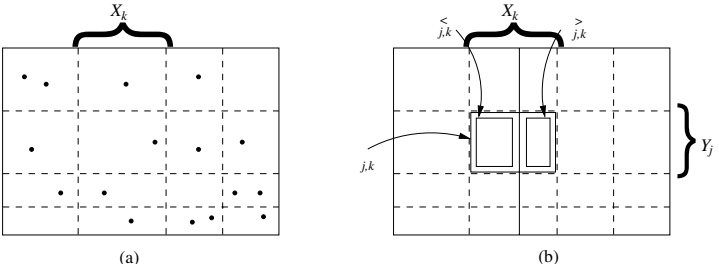


Fig. 2. Finding the median using the grid cells. (a) Slab X_k contains N/t points. (b) $A^<$ and $A^>$ are computed by splitting X_k along ℓ.

c) If u is partitioned using a *rectangle cut*, use the following algorithm to determine the sides of b'. Let ℓ be a line orthogonal to the longest side of b_O that cuts b_O into two fat rectangles and does not intersect b_I. Using only the grid cell counts, decide whether any of the two new regions contains more than $2w(u)/3$ points. If this is the case, then repeat the process in that region. Otherwise, the region with the largest number of points becomes b'. Scan the (up to) four slabs that contain the sides of b' and compute the counts of the "subcells". These counts will be stored in $A^<$, the cell count matrix for $b' \setminus b_I$, and $A^>$, the cell count matrix for $b_O \setminus b'$. Go to 5.(d).

d) Create a new wp-tree node for each of the two regions constructed. For each of these two nodes, determine its partition by repeating step 5, in which the role of A is played by $A^<$ or $A^>$. Stop when reaching level $\log_2 t$.

6. Scan S_x and S_y and distribute the N points into t pairs of sets (S_x^i, S_y^i), corresponding to the t leaves v_i of $\mathcal{T}$.
7. For each pair of sets (S_x^i, S_y^i) do the following. If (S_x^i, S_y^i) fits in memory, then construct the remaining wp-tree nodes. Otherwise, recursively call **Bulk_load** on (S_x^i, S_y^i, v_i).

Theorem 1. *A blocked restricted wp-tree can be bulk loaded in $O(n \log_m n)$ I/Os.*

Proof. First note that sorting the points takes $O(n \log_m n)$ I/Os. Once sorted, the points are kept sorted throughout the recursive calls to the **Bulk_load** procedure. Next note that the choice of $t = \Theta(\min\{m, \sqrt{M}\}) = O(\sqrt{m})$ means that the original $t \times t$ count matrix A fits in memory. In fact, since each of the $2^{\log_2 t} = t$ nodes built in one call to **Bulk_load** adds at most t counts, all count matrices produced during one such call fit in memory.

Now consider one call to **Bulk_load**. Steps 1, 2 and 3 of **Bulk_load** are linear scans of the input sets S_x and S_y using $O(n)$ I/Os. Step 6 can also be performed in $O(n)$ I/Os since S_x and S_y are distributed into $t = \Theta(\min\{m, \sqrt{M}\}) = O(m)$ sets (which means that one block for each of the sets can be maintained in memory during the distribution). Step 5 (recursively) computes a subtree of height $\log_2 t$, using a different algorithm for each of the three partition types.

A geometric or rank cut (Step 5.(a) or 5.(b)) can be computed in $O(|S_x|/t)$ I/Os since slab X_k is scanned at most three times. Similarly, a rectangle cut (Step 5.(c)) can also be computed in $O(|S_x|/t)$ I/Os. The details of this argument will be given in the full paper. It can also be proven that a rectangle cut always exists [7]. Summing up over the $2^{\log_2 t} = O(t)$ nodes built, we obtain that Step 5 can be performed in $O(n)$ I/Os.

Since a subtree of height $\Theta(\log_2 t)$=$\Theta(\log_2 m)$ can be built in a linear number of I/Os (one call to **Bulk_load**), the cost of building the entire blocked restricted wp-tree is $O(n\frac{\log_2 n}{\log_2 m}) = O(n \log_m n)$ I/Os.

Corollary 1. *A kd-tree, BBD-tree, BAR-tree or pseudo-quad-tree can be bulk loaded in $O(n \log_m n)$ I/Os.*

4 The Dynamization Framework

In this section we present a framework for making wp-trees dynamic. We present three methods: the first one takes advantage of the weight balancing property of wp-trees and uses partial rebuilding to maintain the tree balanced [6,21], and the other two methods are based on the so-called logarithmic method [10,21]. All three methods take advantage of the improved bulk loading bounds obtained in the previous section. While the methods are not new, we show how their application to blocked restricted wp-trees produces new dynamic data structures for indexing points in $\mathbb{R}^2$ that are competitive with or better than existing data structures in terms of I/O performance. The choice of method for a given data structure depends on its update and query bounds as well as the application the external structure is to be used in.

4.1 Partial Rebuilding

In the definition of a $(\beta, \delta_0, \kappa)$ wp-tree, the weight condition is satisfied by any $\delta > \delta_0$. This method of relaxing the weight condition allows us to perform updates with good amortized complexity. A node v is said to be *out of balance* if there is another node u such that $u^{(\kappa)} = v$ and $w(u) > \delta w(v)$. In other words, a node is out of balance if one of its descendants is too heavy. A node v is said to be *perfectly balanced* if all nodes u such that $u^{(\kappa)} = v$ satisfy $w(u) \leq \delta_0 w(v)$.

In order to allow dynamic updates on a blocked wp-tree, we employ a partial rebuilding technique, used by Overmars [21] to dynamically maintain quad-trees and kd-trees balanced, and first adapted to external memory by Arge and Vitter [6]. When inserting a new point into the data structure, we first insert it in the appropriate place among the leaves, and then we check for nodes on the path to the root that are out of balance. If v is the highest such node, we rebuild the whole subtree rooted at v into a perfectly balanced tree. In the full paper we prove the following.

Theorem 2. *Let $\mathcal{T}$ be a blocked restricted wp-tree on N points. We can insert points into $\mathcal{T}$ in $O\left(\frac{1}{B}(\log_m n)(\log_2 n) + \log_B n\right)$ I/Os, amortized, and delete points from $\mathcal{T}$ in $O(\log_B n)$ I/Os, worst case. Point queries take $O(\log_B n)$ I/Os, worst case.*

As n goes to infinity, the first additive term dominates the insertion bound. In practice, however, we expect the behavior to be consistent with the second term, $O(\log_B n)$, because the value of B is in the thousands, thus cancelling the effect of the $\log_2 n$ factor in the first term for all practical values of n.

4.2 Logarithmic Methods

The main idea in the logarithmic method [10,21] is to partition the set of input objects into $\log_2 N$ subsets of increasing size 2^i, and build a perfectly balanced data structure $\mathcal{T}_i$ for each of these subsets. Queries are performed by querying each of the $\log_2 N$ structures and combining the answers. Insertion is performed by finding the first empty structure $\mathcal{T}_i$, discarding all structures $\mathcal{T}_j$, $0 \leq j < i$, and building $\mathcal{T}_i$ from the new object and all the objects previously stored in $\mathcal{T}_j$, $0 \leq j < i$. One can adapt the method to external memory by letting the ith subset contain either 2^i *blocks* of points or B^i *points*. We call the two resulting methods *the logarithmic method in base 2* and *the logarithmic method in base B*, respectively.

Logarithmic method in base 2. As mentioned, the ith subset contains 2^i blocks, or $B \cdot 2^i$ points, $0 \leq i \leq \log_2 n$. Queries are performed by combining the answers from the $\log_2 n$ structures. Insertions are performed as in the internal memory case, but we need to maintain a block in internal memory. All insertions go into this block until the block is full, at which time the rebuilding is performed using all points in the block. In the full paper we prove the following.

Theorem 3. *A forest of perfectly balanced blocked restricted wp-trees for indexing N points can be maintained such that insertions take $O\left(\frac{1}{B}(\log_m n)(\log_2 n)\right)$ I/Os, amortized, deletions take $O((\log_m n)(\log_2 n))$ I/Os, worst case, and point queries take $O((\log_B n)(\log_2 n))$ I/Os, worst case.*

Note that, for realistic values of n, m and B, we need less than one I/O, amortized, to insert a point This should be compared to the (at least) $O(\log_B n)$ used in the partial rebuilding method. However, the deletion and point query bounds of this method are worse than the bounds obtained using partial rebuilding.

Logarithmic method in base B. Arge and Vahrenhold used the logarithmic method in base B to obtain an I/O-efficient solution to the dynamic point location problem [5]. Following closely the ideas of Arge and Vahrenhold, we obtain the following.

Theorem 4. *A forest of perfectly balanced blocked restricted wp-trees for indexing N points can be maintained such that insertions take $O((\log_m n)(\log_B n))$ I/Os, amortized, deletions take $O(\log_B n)$ I/Os, amortized, and point queries take $O(\log_B^2 n)$ I/Os, worst case.*

The insertion bound of the base B method is a factor of $\frac{B}{\log_2 B}$ worse than the bound obtained using the base 2 method. The deletion bound, however, is improved by a factor of $\log_2 n$.

4.3 Applications

We now briefly state the results we obtain when using the three dynamization methods on our two running examples, kd-trees and BBD-trees.

The kd-tree. In the full paper we show how we can exploit a property of the kd-tree partitioning method to obtain worst-case bounds on the number of I/Os needed to perform a range query. We obtain the following.

Theorem 5. *Using partial rebuilding, a dynamic external kd-tree can be designed, which answers range queries in $O(\sqrt{N}^{\log_{1/\delta} 2}/\sqrt{B} + K/B)$ I/Os in the worst case, where K is the number of points reported. Each insertion takes $O\left(\frac{1}{B}(\log_m n)(\log_2 n) + \log_B n\right)$ I/Os, amortized, and each deletion takes $O(\log_B n)$ I/Os, worst case. Using the logarithmic method in base 2 (or in base B), a structure with an $O(\sqrt{n} + K/B)$ worst-case range query bound can be designed. In this case insertions take $O\left(\frac{1}{B}(\log_m n)(\log_2 n)\right)$ I/Os, amortized (or $O((\log_m n)(\log_B n))$ I/Os, amortized), and deletions take $O\left((\log_m n)(\log_2 n)\right)$ I/Os, worst case (or $O(\log_B n)$ I/Os, amortized).*

Using the logarithmic methods, the query bound of the dynamic structure is the same as the bound for the static structure, although a logarithmic number of trees are queried in the worst case. This is true in general for a structure with polynomial query bound, because the cost to search each successive structure is geometrically decreasing. If the query bound on the static structure is polylogarithmic (as in our next example), the bound on the dynamic structure increases.

The BBD-tree. The BBD-tree can be used to answer $(1 + \epsilon)$-approximate nearest neighbor queries [7]. Using our dynamization methods we obtain the following.

Theorem 6. *Using partial rebuilding a dynamic external BBD-tree can be designed, which answers a $(1 + \epsilon)$-approximate nearest neighbor query in $Q_{BBD}(N) = O(c(\epsilon)(\log_m n)(\log_2 n)/B + \log_B n)$ I/Os, where $c(\epsilon) = 2\left\lceil 1 + \frac{12}{\epsilon}\right\rceil^2$ [7]. Insertions take $O\left(\frac{1}{B}(\log_m n)(\log_2 n) + \log_B n\right)$ I/Os, amortized, and deletions take $O(\log_B n)$ I/Os, worst case. Using the logarithmic method in base 2 (or in base B), the query bound increases to $Q_{BBD}(N)\log_2 n$ (or $Q_{BBD}(N)\log_B n$). Insertions take $O\left(\frac{1}{B}(\log_m n)(\log_2 n)\right)$ I/Os, (or $O((\log_m n)(\log_B n))$ I/Os), amortized, and deletions take $O\left((\log_m n)(\log_2 n)\right)$ I/Os, worst case (or $O(\log_B n)$ I/Os, amortized).*

References

1. A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31:1116–1127, 1988.
2. L. Arge. External memory data structures. In J. Abello, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of Massive Data Sets*. Kluwer Academic Publishers, 2001. (To appear).
3. L. Arge, K. H. Hinrichs, J. Vahrenhold, and J. S. Vitter. Efficient bulk operations on dynamic R-trees. In *Proc. Workshop on Algorithm Engineering, LNCS 1619*, pages 328–347, 1999.
4. L. Arge, V. Samoladas, and J. S. Vitter. On two-dimensional indexability and optimal range search indexing. In *Proc. ACM Symp. Principles of Database Systems*, pages 346–357, 1999.
5. L. Arge and J. Vahrenhold. I/O-efficient dynamic planar point location. In *Proc. ACM Symp. on Computational Geometry*, pages 191–200, 2000.
6. L. Arge and J. S. Vitter. Optimal dynamic interval management in external memory. In *Proc. IEEE Symp. on Foundations of Comp. Sci.*, pages 560–569, 1996.
7. S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, Nov. 1998.
8. R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1:173–189, 1972.
9. J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.
10. J. L. Bentley. Decomposable searching problems. *Inform. Process. Lett.*, 8:244–251, 1979.
11. D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.
12. M. de Berg, J. Gudmundsson, M. Hammar, and M. Overmars. On R-trees with low stabbing number. In *Proc. Annual European Symposium on Algorithms*, pages 167–178, 2000.
13. C. A. Duncan, M. T. Goodrich, and S. Kobourov. Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 300–309, N.Y., Jan. 17–19 1999. ACM-SIAM.
14. V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
15. R. Grossi and G. F. Italiano. Efficient cross-trees for external memory. In J. Abello and J. S. Vitter, editors, *External Memory Algorithms and Visualization*. American Mathematical Society, 1999.
16. A. Henrich, H.-W. Six, and P. Widmayer. Paging binary trees with external balancing. In *Proc. Graph-Theoretic Concepts in Computer Science, LNCS 411*, pages 260–276, 1989.
17. K. V. R. Kanth and A. K. Singh. Optimal dynamic range searching in non-replicating index structures. In *Proc. International Conference on Database Theory, LNCS 1540*, pages 257–276, 1999.
18. D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading MA, second edition, 1998.
19. D. Lomet and B. Salzberg. The hB-tree: A multiattribute indexing method with good guaranteed performance. *ACM Transactions on Database Systems*, 15(4):625–658, 1990.

20. J. Nievergelt, H. Hinterberger, and K. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, 1984.
21. M. H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes Comput. Sci.* Springer-Verlag, Heidelberg, West Germany, 1983.
22. J. Robinson. The K-D-B tree: A search structure for large multidimensional dynamic indexes. In *Proc. SIGMOD Intl. Conf. on Management of Data*, pages 10–18, 1981.
23. H. Samet. *The Design and Analyses of Spatial Data Structures.* Addison Wesley, MA, 1990.
24. B. Seeger and H.-P. Kriegel. The buddy-tree: An efficient and robust access method for spatial data base systems. In *Proc. International Conf. on Very Large Databases*, pages 590–601, 1990.
25. Y. V. Silva Filho. Average case analysis of region search in balanced k-d trees. *Inform. Process. Lett.*, 8:219–223, 1979.
26. S. Subramanian and S. Ramaswamy. The P-range tree: A new data structure for range searching in secondary memory. In *Proc. ACM-SIAM Symp. on Discrete Algorithms*, pages 378–387, 1995.
27. J. S. Vitter. External memory algorithms and data structures. In J. Abello and J. S. Vitter, editors, *External Memory Algorithms and Visualization*, pages 1–38. American Mathematical Society, 1999.

A Characterization of Temporal Locality and Its Portability across Memory Hierarchies

Gianfranco Bilardi[1⋆] and Enoch Peserico[2⋆⋆]

[1] Dipartimento di Elettronica ed Informatica, Università di Padova, Via Gradenigo 6/A, 35131 Padova, Italy. bilardi@dei.unipd.it
[2] MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA. enoch@theory.lcs.mit.edu

Abstract. This paper formulates and investigates the question of whether a given algorithm can be coded in a way efficiently portable across machines with different hierarchical memory systems, modeled as $a(x)$-HRAMs (Hierarchical RAMs), where the time to access a location x is $a(x)$.
The *width decomposition* framework is proposed to provide a machine-independent characterization of temporal locality of a computation by a suitable set of *space reuse* parameters. Using this framework, it is shown that, when the *schedule*, i.e. the order by which operations are executed, is fixed, efficient portability is achievable. We propose (a) the *decomposition-tree* memory manager, which achieves time within a logarithmic factor of optimal on all HRAMs, and (b) the *reoccurrence-width* memory manager, which achieves time within a constant factor of optimal for the important class of *uniform* HRAMs.
We also show that, when the schedule is considered as a degree of freedom of the implementation, there are computations whose optimal schedule does vary with the access function. In particular, we exhibit some computations for which any schedule is bound to be a polynomial factor slower than optimal on at least one of two sufficiently different machines. On the positive side, we show that relatively few schedules are sufficient to provide a near optimal solution on a wide class of HRAMs.

1 Introduction

In recent years, the importance of the memory hierarchy has grown considerably, and is projected to continue growing in the future, as a result of technological developments [MV99] as well as some fundamental physical constraints [BP97-99]. A number of studies, e.g., [AACS87,ACS87,ACS90,ACFS94,V98,FLPR99], have investigated models of computation that explicitly capture at least some of the hierarchical aspects of modern memory systems, proposing novel algorithms and

⋆ Supported in part by the Italian National Research Council, and by the Italian Ministry of University and Research.
⋆⋆ Supported in part by DARPA under the Air Res. Lab contract F30602-99-2-0511.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 128–139, 2001.

compiler code restructuring techniques [W95] to achieve optimal performance on those models.

Designing efficient algorithms for the memory hierarchy is made difficult by the fact that performance is affected in various ways by the structure of the hierarchy, (e.g., by the number, size, and speed of the various levels): in fact, a priori, the optimal implementation of an algorithm might depend upon it in a way that makes it impossible to achieve optimal performance on different machines with the same code. In the outlined scenario, we formulate the following question: *To what extent can a program be made efficiently portable across a class of machines with different memory hierarchies?*

In [BP00], we outline a general approach toward an analytical formulation of the above question, as a prerequisite for a quantitative study of the issue. Intimately related to the investigation of portability is the machine-independent characterization of those properties of a computation, such as various forms of locality and parallelism, that determine its execution time on a memory hierarchy. Of paramount importance among these properties is *temporal locality* which, informally speaking, allows a computation to be carried out by accessing only a small set of memory locations at a time. In this paper, we substantiate the technical feasibility of the approach proposed in [BP00] by an investigation on the portability of sequential programs across hierarchies where temporal locality is essentially the key factor of performance. We obtain quantitative characterizations of temporal locality and of its portability across memory hierarchies. The focus on sequential temporal locality is justified by its relevance and by the need to gain insights on the general issues in a relatively simple setting. It remains desirable to extend the analysis to include space locality and parallelism.

In Section 2, we define the H-RAMs, the class of target machines considered in our study. They are essentially uniprocessors with a random access memory where an access to memory location x takes time $a(x)$, with $a(x)$ being a generic non-negative, non-decreasing function of x. H-RAMs differ significantly from "real" machines in a number of ways (e.g., they lack block transfer) but we feel they are an excellent model for capturing and isolating the issues of temporal locality. We then define the notion of computation dag, by which we model a computation as a set of operations and their data dependencies. Informally, a computation admits many implementations which differ along two important dimensions: (i) the order of execution of the operations, and (ii) the way data are mapped to memory locations during execution. In the next sections we examine the impact of each dimension on the portability of the implementation.

In Section 3, we assume that the operation schedule has been fixed, and we consider how to best manage memory for that schedule. First, we introduce the key notion of W*-width decomposition* of a schedule τ and the corresponding parameter $r_\tau(W)$, called the *space reuse*, which informally corresponds to the number of subcomputations, each using approximately W space, into which τ can be partitioned. The parameters $r_\tau(2^\ell)$, (where $\ell = 0, \ldots, \lceil \log S \rceil$ and S is the minimum space required to execute τ) are sufficient to characterize the optimal execution time of τ on a wide class of HRAMs, wielding the first quantitative

characterization of the qualitative notion of *temporal locality*. We provide lower bounds matched, within a logarithmic factor, by the *decomposition-tree* memory management strategy, and within a constant factor on a wide class of machines, by the *reoccurrence-width* strategy. Neither of these strategies takes into account the access function $a(x)$, indicating that, for a fixed operation schedule, efficient portability is indeed achievable.

In Section 4, we turn our attention to the impact of the operation schedule. Several interesting cases of computations (such as some algorithms for FFT, matrix multiplication, and sorting) are known to admit optimal schedules for classes of uniform HRAMs [AACS87,FLPR99]. However, our main findings are on the negative side: by developing ideas formulated in [P98], we show that, at least for some computations, the optimal schedule is different on different machines, both in the case of *recomputation* (multiple executions of the same operations are allowed) and of *no recomputation*. In each case we also provide lower bounds to the loss of time performance that any schedule suffers on at least one machine of a reasonable class. These results require a novel approach to analyze tradeoffs between the number of accesses in different regions, tradeoffs not captured by previous lower bound techniques for the memory hierarchy [HK81,S97,BP97-99,BPD00].

One consequence of our results is that, to obtain generally portable code, such code must be somehow parametrized with parameters adaptable to those of the memory hierarchy, either statically, by a compiler, or dynamically, like in the systems FFTW [FJ98], ATLAS [WD], and PHiPAC [BACD97]. While we take some preliminary steps to estimate the size of the parameter space of the code as a function of the acceptable loss of performance, this remains largely an uncharted and promising territory for further investigations.

2 Models of Machines and Computations

We shall model machines as *Hierarchical Random Access Machines* (HRAMs), a model very close to the HMM of [AACS87] and to the H-RAM of [BP97-99]. An HRAM consists of a serial processor, a program memory, and a data memory. Both memories are random access and consist of locations with addresses ranging over the nonnegative integers. Program memory is assumed to be very fast, with access time subsumed within the processor cycle. Data memory is hierarchical, with an access to address x taking $a(x)$ units of time. The *access function* $a(\cdot)$ satisfies $0 \leq a(x) \leq a(x+1)$, for any $x \geq 0$. To stress the role of the access function, we shall use the notation $a(x)$-HRAM for the machine. It is also quite useful to introduce the *cumulative access function* $A(x) = \sum_{y=0}^{x-1} a(y)$, for $x \geq 1$, with $A(0) = 0$.

As it emerged since some of the early investigations [ST85,AACS87], comparison between upper and lower bounds on HRAM execution time often leads to consideration of the ratio $a(y)/a(x)$ of access times for a suitably bounded value of the ratio y/x of the corresponding addresses, motivating the following definition.

Definition 1. *For given real parameters $\xi, \alpha \geq 1$, an access function $a(x)$ or the corresponding $a(x)$-HRAM is said to be* (ξ, α)-uniform *if $a(\xi x) \leq \alpha a(x)$, for any x.*

We now turn our attention to computations, which we model as computation dags ($+$ and Σ denote the union operation restricted to disjoint sets).

Definition 2. *A* computation directed acyclic graph *(CDAG) is a 4-tuple $C = (I, V, E, O)$ of finite sets such that: (1) $I \cap V = \emptyset$; (2) $E \subseteq (I + V) \times V$ is the* set of arcs*; (3) $G = (I + V, E)$ is a* directed acyclic graph *with* no *isolated vertices; (4) I is called the* input set*; (5) V is called the* operation set *and all its vertices have* one or two *incoming arcs; (6) $O \subseteq I + V$ is called the* output set.

Informally, with each vertex in $I + V$ we associate a value. For a vertex in I, the value is externally supplied and hence considered an *input* to the computation. For a vertex in V, the value is the *result of an operation* whose operands are provided by the predecessors of that vertex. We have restricted the number of operands to be one or two. The set O defines which values, among all the ones being input or computed, form the desired *output* set. The main advantage of the CDAG model for the present investigation is that it specifies neither the order in which the operations have to be executed nor the memory locations where data have to be stored, which we consider degrees of freedom for the implementation. Furthermore, hierarchy related space complexity issues have been extensively investigated using the CDAG model ([S97]).

3 Memory Management for a Fixed Operation Schedule

In this section, we consider a computation modeled by a CDAG $C = (I, V, E, O)$, together with a given feasible schedule τ of its operations. For simplicity, we develop the analysis for a schedule without recomputation, modeled as a topological sorting $\tau = (v_1, \ldots, v_N)$ of the elements of V, that is, whenever $(v_i, v_j) \in E$, then $i < j$. All the results of the present section are readily generalized to schedules with recomputation.

Given τ, a program computing C must still choose in which memory locations, over the course of the computation, the values involved will be stored, the objective being a minimization of the running time over HRAMs with as wide a range of access functions $a(x)$ as possible.

We begin by studying some lower limits that any memory map has to satisfy for a given schedule.

3.1 Lower Bounds

The intuition that, at any given time, at least one memory location must be used for each value already available and yet needed leads to the following definition and to the subsequent propositions.

Definition 3. *For $0 \leq i \leq N$, the* width at *i of a schedule $\tau = (v_1, \ldots, v_N)$ of a CDAG $C = (I, V, E, O)$ is the quantity $W_\tau(i) = |Z_\tau(i)|$, where*

$$Z_\tau(i) = \{u \in I \cup \{v_1, \ldots, v_i\} : \ (u \in O) \vee (\exists j > i : (u, v_j) \in E)\}.$$

The width *of a schedule $\tau = (v_1, \ldots, v_N)$ is the quantity $W_\tau = \max_i W_\tau(i)$.*

Next, we begin to develop the relationship between width, space, and time.

Proposition 1. *The amount S_τ of space needed by any HRAM execution of CDAG C according to schedule τ satisfies $S_\tau \geq W_\tau$.*

Proposition 2. *The time T_τ needed by an $a(x)$-HRAM execution of CDAG C according to schedule τ satisfies $T_\tau \geq \sum_{h=0}^{W_\tau - 1} a(h) = A(W_\tau)$.*

The analysis embodied by Propositions 1 and 2 can be refined by partitioning a computation of C into contiguous subcomputations and considering their separate accesses. Next, we review the notion of *topological partition* [BP97-99] and we formalize the notion of *subcomputation*.

Definition 4. *The sequence $(V_1, V_2, \ldots, V_q)$ of nonempty subsets of V is called a* topological partition *of CDAG $C = (I, V, E, O)$ if $V = \sum_{h=1}^{q} V_h$ and $E \subseteq (\cup_{h=1}^{q} \cup_{k=h+1}^{q} (I + V_h) \times V_k)$.*

Definition 5. *Let $(V_1, V_2, \ldots, V_q)$ be a topological partition of CDAG $C = (I, V, E, O)$. Then, for $h = 1, 2, \ldots, q$, the* subcomputation *of C induced by V_h is the CDAG $C_h = (I_h, V_h, E_h, O_h)$, where:*

$$\begin{aligned} I_h &= \{u \notin V_h : \ \exists v \in V_h \text{ s.t. } (u, v) \in E\}, \\ O_h &= \{u \in I_h + V_h : \ u \in O \vee (\exists v \notin V_h \text{ s.t. } (u, v) \in E)\}, \\ E_h &= E \cap ((I_h + V_h) \times V_h). \end{aligned}$$

Proposition 3. *With the notation of Definition 5, let τ_h be a schedule for subcomputation C_h, for $h = 1, 2, \ldots, q$, and let the concatenation $\tau =< \tau_1, \ldots, \tau_q >$ be the corresponding schedule for C. Then, for any $S \geq 0$, the number $Q(S)$ of accesses to locations with address $x \geq S$ made by any HRAM execution of C according to schedule τ satisfies $Q(S) \geq \sum_{h=1}^{q} (W_{\tau_h} - S)$.*

Proposition 4. *With the notation of Proposition 3, the running time of any execution of C according to schedule τ on an $a(x)$-HRAM satisfies*

$$T_\tau \geq \sum_{h=1}^{q} \sum_{i=0}^{W_{\tau_h} - 1} a(i) = \sum_{h=1}^{q} A(W_{\tau_h}). \tag{1}$$

Of particular interest for our developments are topological partitions where all the subcomputations, except possibly for the last one, have nearly the same width. These partitions, to be defined next, will enable us to derive tight lower and upper bounds on computation time.

Definition 6. *Given a CDAG $C = (I, V, E, O)$, we say that $< \tau_1, \ldots, \tau_q >$ is the* W-width decomposition *of a schedule τ for C if (i) $\tau =< \tau_1, \ldots, \tau_q >$, (ii) $W_{\tau_h} \leq W$ for $h = 1, 2, \ldots, q$, and (iii) $W_{\tau_h^+} > W$ for $h = 1, 2, \ldots, q-1$, where τ_h^+ is obtained by appending to τ_h the next operation in τ.*

All subcomputations in a W-width decomposition, except possibly for the last one, are *full* in the sense that their width is at least $W - 2$. The number $r_\tau(W) \in \{q-1, q\}$ of full subcomputations, hereafter referred to as the *space reuse* (at W), gives the following valuable information on the access complexity:

$$Q_\tau(W/2 - 2) \geq (W/2) r_\tau(W). \tag{2}$$

This bound follows easily from Proposition 3 by setting $S = W/2-2$, $q = r_\tau(W)$, and $W_{\tau_h} - S = W - (W/2-2) > W/2$. Let H be the smallest integer greater than two such that $W_\tau \leq 2^H$. For $2 < \ell < H$, let $n_\ell = r_\tau(2^\ell)$ be the number of full subcomputations in the 2^ℓ-width decomposition of τ. Also, let $2n_2 = |E| + |V|$. It turns out that n_ℓ's closely characterize the temporal locality properties of a schedule, as the next result shows from the lower bound perspective.

Theorem 1. *Let τ be a schedule for CDAG C and let n_ℓ be defined as above, for $\ell = 2, 3, \ldots, H$. Then, the time of any execution of C according to schedule τ on an $a(x)$-HRAM satisfies the bound*

$$T_\tau \geq \sum_{\ell=2}^{H-1} (n_\ell - 2n_{\ell+1}) 2^{\ell-1} a(2^{\ell-1} - 2). \tag{3}$$

3.2 Efficient Strategies for Memory Management

We now turn our attention to constructive ways of defining a memory map for a given schedule $\tau = (v_1, v_2, \ldots, v_N)$ of computation C. It can be easily shown that the lower bound given by Proposition 1 is tight, i.e., $S_\tau = W_\tau$. However, when the objective is the minimization of execution time on an H-RAM, it is crucial not only to reduce the overall space, but also to bias the utilization of space toward the smaller, faster locations. We have developed two memory managers. The first, named the *decomposition-tree* memory manager, uses a variant of the topological-separator proposed in [BP97-99] that yields a tighter control of space, essential for our present purposes.

Decomposition-Tree Memory Management

1. Partition τ into two subschedules τ_0 and τ_1.
2. Reorganize the inputs of τ so that the inputs of τ_0 lie in the address range $[0, ..., W_{\tau_0} - 1]$, and the remaining inputs of τ lie in the range $[W_{\tau_0}, ..., W_\tau - 1]$.
3. Recursively execute τ_0 within the address range $[0, ..., (W_{\tau_0} - 1)]$.
4. Reorganize data in memory so that the inputs of τ_1 lie in the address range $[0, ..., W_{\tau_1} - 1]$, and the remaining data in the range $[W_{\tau_1}, ..., W_\tau - 1]$.
5. Recursively execute τ_1 within the address range $[0, ..., (W_{\tau_1} - 1)]$.

The formal description of the algorithm and its analysis we leave for the final version of the paper. They lead to the following result:

Theorem 2. *The* balanced, binary *decomposition-tree strategy for memory management yields an $a(x)$-HRAM program P which correctly executes CDAG $C = (V, E, I, O)$ with schedule τ in optimal space $S_\tau = W_\tau$ and time*

$$T_\tau \leq (4\lceil \log N \rceil + 2) T_\tau^{opt}, \tag{4}$$

where T_τ^{opt} denotes the minimum time achievable on an $a(x)$-HRAM.

Theorem 2 makes no restrictive assumption on the access function $a(x)$ modeling HRAM delays. Independence of the access function poses rather stringent constraints on the memory manager, as it implies that no address $x \geq W_\tau = S_\tau$ can be used (otherwise, there would be no way to relate the corresponding access to T_τ^{opt}). However, stronger results can be obtained if we restrict our attention to uniform HRAMs, which are both physically and technologically sound. In this direction, we develop an alternate memory management strategy which achieves optimality to within a constant factor on uniform HRAMs. Unlike the tree-decomposition approach, the strategy presented below relocates data in memory only in correspondence with operations that involve those data, at an address (approximately) proportional to the amount of space needed by the subcomputation intervening between the current operation and the next operation where that value occurs as an operand.

Let H be the integer such that $2^{H-1} < W_\tau \leq 2^H$. We define memory region $M_2 = \{0, 1\}$ and, for $\ell = 3, \ldots, H$, memory region $M_\ell = \{2^\ell - 6, \ldots, 2^{\ell+1} - 7\}$, of size 2^ℓ.

We shall say "*store in region M_ℓ*" as an abbreviation for "*store in the location of smallest address among those currently available in M_ℓ.*" We shall also use the shorthand $w_\tau(u|@v_k)$ to indicate the width of the subcomputation between the two consecutive occurrences v_i and v_j of u, with $i \leq k < j$.

Reoccurrence-Width Memory Management

1. *Input.* We assume that the inputs $u_0, u_1, \ldots, u_{|I|-1}$ are originally available at the $|I|$ lowest memory locations. For each $h = |I|-1, \ldots, 1, 0$, move input u_h to the smallest region of size at least $w_\tau(u|@v_0)$, excluding M_2.
2. *Operations.* For $k = 1, 2, \ldots, N$, do:
   (i) Load: Move the operand(s) of v_k to location 0 (and 1), in M_2.
   (ii) Compute: Execute operation v_k storing the result in M_ℓ, with $\ell = \max(3, \lceil \log w_\tau(v_k|@v_k) \rceil)$.
   (iii) Relocate: Move each operand u of v_k subsequently needed in the computation to M_ℓ, with $\ell = \max(3, \lceil \log w_\tau(u|@v_k) \rceil)$.
3. *Output.* The outputs $z_0, z_1, \ldots, z_{|O|-1}$ are now stored in non decreasing order of the quantity $\ell = \max(3, \lceil \log w_\tau(z_h|@v_{N+1}) \rceil)$. For each $h = 0, 1, \ldots, |O| - 1$, move output z_h to location h.

A somewhat complex analysis establishes the correctness of the above strategy and provides an upper bound to the resulting HRAM running time, in terms of the access function $a(x)$ (and the related $A(x) = \sum_{y=0}^{x-1} a(x)$) characterizing

the machine and in terms of the parameters n_ℓ's characterizing the computation. The resulting upper bound is also compared to the lower bound of Theorem 1, thus establishing the optimality, to within a constant factor, of the reoccurrence-width strategy, for a wide class of access functions.

Theorem 3. *The reoccurrence-width strategy yields an $a(x)$-HRAM program P which executes CDAG $C = (V, E, I, O)$ with schedule τ in time*

$$T_\tau \leq \sum_{\ell=2}^{H-1} (n_\ell - 2n_{\ell+1}) 2^{\ell+1} a(2^{\ell+2} - 7) + A(|I|) + A(|O|). \tag{5}$$

If T_τ^{opt} denotes the minimum time achievable on an $a(x)$-HRAM, we have:

$$T_\tau \leq (4\gamma(W_\tau) + 1) T_\tau^{opt}. \tag{6}$$

where $\gamma(W_\tau) = \max_{\ell \in \{3,\ldots,H-1\}} (a(2^{\ell+2} - 7)/a(2^{\ell-1} - 2))$.

For most functions $a(x)$ of interest, the quantity $a(8x + 9)/a(x)$, and hence the quantity $\gamma(W_\tau)$ in Relation 6, is bounded above by a constant. E.g., if $a(x)$ is $(25/2, \alpha)$-uniform, then $\gamma(W_\tau) \leq \alpha$. In the interesting case where $a(x) = a_0\sqrt{x+1}$ modeling speed-of-light delays in planar layouts, we obtain $\gamma(W_\tau) \leq \sqrt{26/3} < 3$.

4 Optimal Schedule and Memory Access Function

Some CDAGs, such as those corresponding to matrix multiplication and of the radix-2 FFT, do admit a portable schedule that is simultaneously (near) optimal for all machines (in a wide class); it is natural to ask whether any CDAG does [P98,FLPR99]. Below, we answer this question negatively by exhibiting CDAGs for which it can be shown that there are HRAM pairs such that, for any schedule, at least on one of the two machines, time performance is significantly suboptimal, i.e. by a factor polynomial (with exponent between 0 and 1) in the number N of operations. It is quite possible that a given CDAG admits a portable schedule if recomputation is allowed but it does not when recomputation is disallowed, and vice versa. For this reason, we deal with the two cases separately.

Theorem 4. *Let M_0 denote a 1-HRAM (the standard RAM) and $M_{1/2}$ denote a $\sqrt{x}$-HRAM. For infinitely many N, there exists a CDAG $C(N)$ of N operations such that the running time of any schedule, allowing* recomputation*, is suboptimal by a factor $\Omega(N^{1/6})$ on M_0 or on $M_{1/2}$.*

Space limitations force us once more to leave the proof to the full version of the paper, but we will attempt to convey an idea of the line of argument. The CDAG $C(N)$ is actually the member $R^{N^{1/6}}_{N^{5/6}}$ of a family whose generic element R^r_c has one input and $N = rc - 1$ operation nodes. The nodes of R^r_c are connected as a linear chain which can be visualized as folded into r rows of c columns each, with additional arcs connecting nodes that are consecutive in a column. Figure 1 illustrates R^4_6. Intuitively, on machine M_0, the best schedule is the one that performs no recomputation. On machine $M_{1/2}$, instead, a better schedule computes

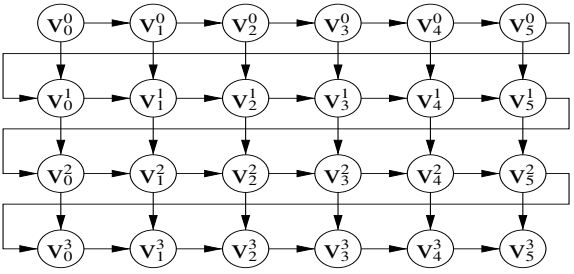


Fig. 1. The CDAG R_6^4

.

node v_j^i by first recomputing the entire column above it from the column above the previous node (and from v_0^0 if $j = 0$), using only $O(r)$ memory locations for the whole computation. Our analysis shows that the class of all schedules can be partitioned into two subclasses depending on whether (according to a suitable technical measure) the amount of recomputation being performed is above or below a certain threshold. It is then shown that a schedule above (respectively, below) the threshold is bound to incur a significant time loss on M_0 (respectively, $M_{1/2}$). We observe that R_c^r can be viewed as the CDAG describing the processing performed during $N - 1$ steps by a simple type of digital filter of order c. It appears quite likely that similar behaviours will be exhibited by other natural CDAGs. A similar result can be obtained when ruling out recomputation:

Theorem 5. *Let M_α denote an x^α-HRAM. Given any pair α and β of real numbers with $0 \leq \alpha < \beta < 1$, for infinitely many N, there exists a CDAG $C_{\alpha,\beta}(N)$ of N operations such that the running time of any schedule $\tau_{\alpha,\beta}(N)$ with* no recomputation *is suboptimal by a factor $\Omega(N^{\frac{\alpha\beta(\beta-\alpha)}{24}})$ on M_α or on M_β.*

The CDAG $C\alpha, \beta(N)$ referred to in the statement of Theorem 5, belongs to a family of CDAGs $G_{m,n}^p$ an element of which is illustrated in Figure 2. Informally, $G_{m,n}^p$ consists of p almost disjoint isomorphic subgraphs, only sharing a set of n inputs $i_0, ..., i_{n-1}$. Subgraph h has a backbone consisting of a long chain:

$$(j^h, u_0^h, ..., u_{n-1}^h, v_0^h, ..., v_{mn-1}^h, w_{n-1}^0, ..., w_{n-1}^h),$$

with j^h as an input and w_{n-1}^h as the designated output. In addition, v_{k-1}^h takes $i_{k\%n}$ as an input, and w_k^h takes u_{n-k}^h as an input. For convenience, we shall refer to the three portions of the operation chain as to the the u-chain, the v-chain, and the w-chain, respectively.

A key property is that, between the computation of the first and of the last vertex of each v-chain, all the values of the corresponding u-chain must reside in memory, since they will be needed for the computation of the w-chain. Then, the following tradeoffs arises: as the number of v-chains simultaneously under execution increases, the space and time to store and retrieve the corresponding u-chains increases, while the time to access the i inputs can be made to decrease, since - once accessed - such an input can be used to advance the computation of

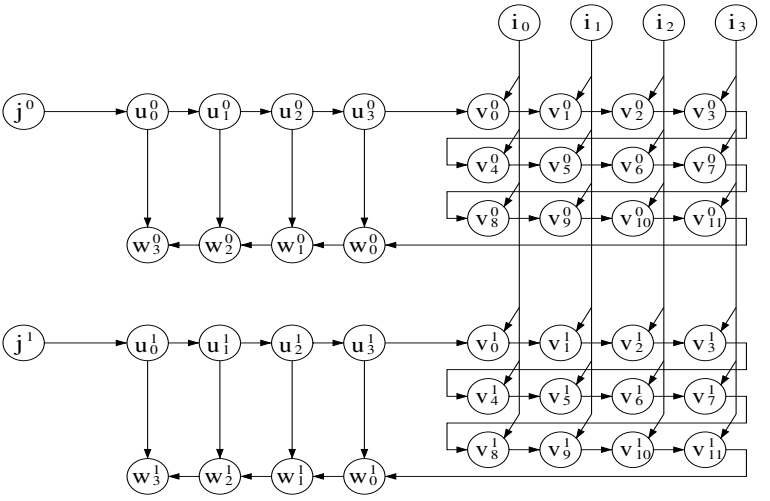


Fig. 2. The CDAG $G^2_{3,4}$

.

several v-chains. Ultimately, it turns out that the optimal point in this tradeoff changes with the access function of the HRAM, with the optimal number of v-chains under simultaneous execution decreasing for machines with "steeper" access functions. The systematic, quantitative analysis of the above tradeoff is rather subtle and left for the full version of the paper.

Theorems 4 and 5 do not rule out the possibility of a parametrized program which, when the parameters are selected (perhaps by a knowledgeable compiler or run-time system) as a function of $a(x)$ achieves optimal performance. The next results explore the question of how many different schedules have to be generated, as the parameters span over their range, in order to achieve a performance within a given factor from optimal on any machine from a suitable class.

Definition 7. *A set of schedules $\mathcal{T} = \{\tau_1, \ldots, \tau_r\}$ of a given a CDAG C is said to be s-optimal with respect to a class $\mathcal{H}$ of HRAMs if, for any $M \in \mathcal{H}$, there is a $\tau \in \mathcal{T}$ such that C can be executed on M with schedule τ in time within a factor s of optimal.*

Theorem 6. *Let $\mathcal{H}(\xi, s)$ be the class of the $(\xi, \sqrt{s})$-uniform HRAMs. Let $\mathcal{C}(\xi, s)$ be the set of CDAGs C such that no optimal schedule of C on any HRAM in $\mathcal{H}(\xi, s)$ requires more than N space. Then we have:*

- *(Upper Bound.) For any $C \in \mathcal{C}(\xi, s)$, there is a set $\mathcal{T}$ of schedules s-optimal with respect to $\mathcal{H}(\xi, s)$ with size $|\mathcal{T}| \leq 2^{\lceil log_\xi N \rceil} \approx N^{1/\log \xi}$.*
- *(Lower Bound.) There is a constant K such that, for any $s \geq 1$, there is an infinite sequence of CDAGs $C(N)$ in $\mathcal{C}(\xi, s)$ of N operations such that any set of schedule $\mathcal{T}$ s-optimal with respect to $\mathcal{H}(\xi, s)$ has size $|\mathcal{T}| \geq N^{K/(\log s + \log\log N)}$, for any ξ.*

From Theorem 6 we see that, although a CDAG might well admit an exponential number of schedules, a suitable polynomial subset of them always contains one with performance within a constant factor of optimal. We remark that the proof of the preceding upper bound result (left to the full version of the paper) exploits approximability properties of the access function $a(x)$, making no use of any special structure of the relevant CDAGs. We also show that, at least for some CDAGs, this number of schedules can not be substantially reduced. The detailed description and analysis of such CDAGs is again left to the full version, but the key idea is to consider CDAGs composed by a family of $\Theta(\log N/(\log s + \log\log N))$ sets of graphs of the type introduced in Theorems 4 and 5. By careful tuning of the parameters, we can make the asymptotic time requirements of the execution of different sets on an $a(x)$-HRAM depend only on the behaviour of $a(x)$ on different, disjoint intervals of addresses, forcing in turn, if s-optimality is to be achieved, a different schedule of the global CDAG according to whether $a(x)$ is sufficiently "steep" or not on each interval. The dependence of the size of s-optimal sets of schedules upon the structure of the CDAG is a very interesting problem and certainly deserves investigation.

5 Conclusions

We have proposed the width framework leading to a quantitative definition of temporal locality which enables performance estimates for an algorithm on different hierarchical systems. Then, we have explored the efficient portability of a fixed implementation of an algorithm on the spectrum of different systems. We have found that the exploitation of the inherent temporal locality of an algorithm through the memory management is quite amenable to a machine-independent optimization. Instead, the optimization of the operation schedule generally requires some knowledge of the target memory system.

This work can be extended in several directions. More general memory models need to be considered, to include block transfer, pipelined accesses, and parallel memories. Indeed, the width framework has already proven useful in the investigation of sequential, pipelined hierarchies [BEP01]. More flexible models of portability are also of interest, where the code is allowed to somehow adapt to the machine. The previous section touches on such issues, but a systematic investigation of "parametrized" algorithms remains desirable. A final observation, for which we are indebted to Peter M. Kogge, is that several of our results could be reinterpreted by viewing $a(x)$ as the energy required to retrieve the content of location x, a metric of interest in the context of low power computing.

References

[AACS87] A. Aggarwal, B. Alpern, A.K. Chandra, M. Snir. A Model for Hierarchical Memory. *Proc. 19th ACM Symp. Theory of Computing.*, (1987), 305–314.

[ACS87] A. Aggarwal, A.K. Chandra and M. Snir. Hierarchical Memory with Block Transfer. *Proc. 28th Symp. Foundations of Comp. Science*, (1987), 204–216.

[ACFS94] B. Alpern, L. Carter, E. Feig and T. Selker. The Uniform Memory Hierarchy Model of Computation. *Algorithmica*, vol. 12, (1994), 72-129.

[ACS90] A. Aggarwal, A.K. Chandra and M. Snir. Communication Complexity of PRAMs. *Theoretical Computer Science*, vol.71, 3-28, 1990.

[BACD97] J.Bilmes, K.Asanovic, C.Chin and J.Demmel. Optimizing matrix multiply using PHiPAC: a portable, high-performance, Ansi C coding methodology. *International Conference on Supercomputing*, (1997).

[BEP01] G.Bilardi, K.Ekanadham, P.Pattnaik Computational power of pipelined memory hierarchies. *13th ACM Symp. Par. Algorithms and Architectures*, (2001).

[BP00] G.Bilardi, E.Peserico. An Approach towards an Analytical Characterization of Locality and its Portability. *Proc. of International Workshop on Innovative Architectures 2000*, IEEE CS Press, (2001).

[BP97-99] G.Bilardi, F.Preparata Processor-time tradeoffs under bounded-speed message propagation. Part I: Upper bounds. *Theory of Computing Systems*, vol. 30, 523-546, 1997. Part II: Lower bounds. *Theory of Computing Systems*, vol. 32, 531-559, 1999.

[BPD00] G. Bilardi, A. Pietracaprina, and P. D'Alberto. On the space and access complexity of computation dags. *26th Workshop on Graph-Theoretic Concepts in Comp. Science* (2000).

[FJ98] M.Frigo and S.G.Johnson. FFTW: An Adaptive Software Architecture for the FFT. *ICASSP*, (1998), 1381-1384.

[FLPR99] M. Frigo, C.E. Leiserson, H. Prokop and S. Ramachandran. Cache-Oblivious Algorithms. *Proc. 40th Symp. Foundations of Comp. Science*, (1999).

[HK81] J.W. Hong and H.T. Kung. I/O Complexity: The Red–Blue Pebble Game. *Proc. 13th ACM Symp. Theory of Computing*, (1981), 326–333.

[HP96] J.L. Hennessy and D.A. Patterson, *Computer Architecture A Quantitative Approach*. Morgan Kaufmann, San Mateo, CA, 1996.

[MV99] V. Milutinovic and M. Valero (Guest Eds.) Special Issue on Cache Memory and Related Problems. *IEEE Transactions on Computers*, 1999.

[P90] S.S. Przybylski. *Cache and Memory Hierarchy Design. A Performance Directed Approach.* Morgan Kaufmann Publishers, Inc. Palo Alto, CA 1990.

[P98] E. Peserico. Performance Optimization on Hierarchical Memory. *Laurea Thesis*, Dip. di Elettronica ed Informatica, Università di Padova, July 1998.

[S97] J.E. Savage. *Models of Computation. Exploring the Power of Computing* Addison-Wesley, Reading, MA, 1998.

[ST85] D.D. Sleator and R.E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, vol. 28(2) 202-208, 1985.

[V98] J.S. Vitter. External Memory Algorithms. Invited paper in *Proc. 6th European Symp. on Algorithms*, (G.Bilardi et al. Eds.), Springer Verlag, (1998), 1–25.

[W95] M.Wolfe High Performance Compilers for Parallel Computing. Addison-Wesley, 1995.

[WD] R.C.Whaley and J.J.Dongarra. Automatically Tuned Linear Algebra Software. http://www.netlib.org/atlas/index.html

The Complexity of Constructing Evolutionary Trees Using Experiments

Gerth Stølting Brodal[1,⋆], Rolf Fagerberg[1,⋆],
Christian N.S. Pedersen[1,⋆], and Anna Östlin[2,⋆⋆]

[1] BRICS[†], Department of Computer Science, University of Aarhus, Ny Munkegade, DK-8000 Århus C, Denmark. {gerth,rolf,cstorm}@brics.dk
[2] Department of Computer Science, Lund University, Box 118, S-221 00 Lund, Sweden. Anna.Ostlin@cs.lth.se

Abstract. We present tight upper and lower bounds for the problem of constructing evolutionary trees in the experiment model. We describe an algorithm which constructs an evolutionary tree of n species in time $O(nd \log_d n)$ using at most $n\lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1} n + O(1))$ experiments for $d > 2$, and at most $n(\log n + O(1))$ experiments for $d = 2$, where d is the degree of the tree. This improves the previous best upper bound by a factor $\Theta(\log d)$. For $d = 2$ the previously best algorithm with running time $O(n \log n)$ had a bound of $4n \log n$ on the number of experiments. By an explicit adversary argument, we show an $\Omega(nd \log_d n)$ lower bound, matching our upper bounds and improving the previous best lower bound by a factor $\Theta(\log_d n)$. Central to our algorithm is the construction and maintenance of separator trees of small height, which may be of independent interest.

1 Introduction

The evolutionary relationship for a set of species is commonly described by an evolutionary tree, where the leaves correspond to the species, the root corresponds to the most recent common ancestor for the species, and the internal nodes correspond to the points in time where the evolution has diverged in different directions. The evolutionary history for a set of species is rarely known, hence estimating the true evolutionary tree for a set of species from obtainable information about the species is of great interest. Estimating the true evolutionary tree computationally requires a model describing how to use available information about species to estimate aspects of the true evolutionary tree. Given a model, the problem of estimating the true evolutionary tree is often referred to as constructing the evolutionary tree in that model.

⋆ Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).
⋆⋆ Partially supported by TFR grant 1999-344.
† Basic Research in Computer Science, www.brics.dk, funded by the Danish National Research Foundation.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 140–151, 2001.

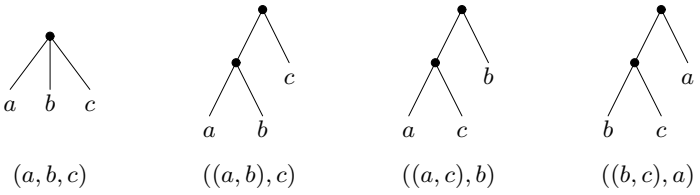


Fig. 1. The four possible outcomes of an experiment for three species a, b and c

In this paper we study the problem of constructing evolutionary trees in the *experiment model* proposed by Kannan, Lawler and Warnow in [6]. In this model the information about the species is obtained by *experiments* which can yield the evolutionary tree for any triplet of species, cf. Fig. 1. The problem of constructing an evolutionary tree for a set of n species in the experiment model is to construct a rooted tree with no unary internal nodes and n leaves labeled with the species such that the topology of the constructed tree is consistent with all possible experiments involving the species. Hence, the topology of the constructed tree should be such that the induced tree for any three species is equal to the tree returned by an experiment on those three species.

The relevance of the experiment model depends on the possibility of performing experiments. A standard way to express phylogenetic information is by a distance matrix. A distance matrix for a set of species is a matrix where entry M_{ij} represents the evolutionary distance between species i and j, measured by some biological method (see [6] for further details). For three species a, b and c where $M_{ab} < \min\{M_{ac}, M_{bc}\}$ it is natural to conclude that the least common ancestor of a and b is below the least common ancestor of a and c, i.e. the outcome of an experiment on a, b and c can be decided by inspecting M_{ab}, M_{ac} and M_{bc}. The consistency of experiments performed by inspecting a distance matrix depends entirely on the distance matrix. Kannan *et al.* in [6] define a distance matrix as noisy-ultrametric if there exists a rooted evolutionary tree such that for all triplets of species a, b and c it holds that $M_{ab} < \min\{M_{ac}, M_{bc}\}$ if and only if the least common ancestor of a and b is below the least common ancestor of a and c in the rooted evolutionary tree. Hence, if a noisy-ultrametric distance matrix for the set of species can be obtained, it can be used to perform experiments consistently. Another and more direct method for performing experiments is DNA-DNA hybridization as described by Sibley and Ahlquist in [9]. In this experimental technique one measures the temperature at which single stranded DNA from two different species bind together. The binding temperature is correlated to the evolutionary distance, i.e. by measuring the binding temperatures between DNA strands from three species one can decide the outcome of the experiment by deciding which pair of the three species bind together at the highest temperature.

Kannan *et al.* introduce and study the experiment model in [6] under the assumption that experiments are flawless in the sense that they do not contradict

each other, i.e. it is always possible to construct an evolutionary tree for a set of species that is consistent with all possible experiments involving the species. They present algorithms for constructing evolutionary trees with bounded as well as unbounded degree, where the degree of a tree is the maximum number of children for an internal node. For constructing binary evolutionary trees they present three different algorithms with running times $O(n \log n)$, $O(n \log^2 n)$ and $O(n^2)$ respectively, using $4n \log n$, $n \log_{3/2} n$ and $n \log n$ experiments respectively, where $\log n$ denotes $\log_2 n$. For constructing an evolutionary tree of degree d they present an algorithm with running time $O(n^2)$ using $O(dn \log n)$ experiments. Finally, for the general case they present an algorithm with running time $O(n^2)$ using $O(n^2)$ experiments together with a matching lower bound. Kao, Lingas, and Östlin in [7] present a randomized algorithm for constructing evolutionary trees of degree d with expected running time $O(nd \log n \log \log n)$. They also prove a lower bound $\Omega(n \log n + nd)$ on the number of experiments. The best algorithm so far for constructing evolutionary trees of degree d is due to Lingas, Olsson, and Östlin, who in [8] present an algorithm with running time $O(nd \log n)$ using the same number of experiments.

In this paper we present the first tight upper and lower bounds for the problem of constructing evolutionary trees of degree d in the experiment model. We present an algorithm which constructs an evolutionary tree for n species in time $O(nd \log_d n)$ using at most $n\lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1} n + O(1))$ experiments for $d > 2$, and at most $n(\log n + O(1))$ experiments for $d = 2$, where d is the degree of the constructed tree. The algorithm is a further development of an algorithm from [8]. Our construction improves the previous best upper bound by a factor $\Theta(\log d)$. For $d = 2$ the previously best algorithm with running time $O(n \log n)$ had a bound of $4n \log n$ on the number of experiments. The improved constant factors on the number of experiments are important because experiments are likely to be expensive in practice, cf. Kannan *et al.* [6]. By an explicit adversary argument, we show an $\Omega(nd \log_d n)$ lower bound, matching our upper bounds and improving the previous best lower bound by a factor $\Theta(\log_d n)$.

Our algorithm also supports the insertion of new species with a running time of $O(md \log_d(n + m))$ using at most $m\lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1}(n + m) + O(1))$ experiments for $d > 2$, and at most $m(\log(n+m) + O(1))$ experiments for $d = 2$, where n is the number of species in the tree to begin with, m is the number of insertions, and d is the maximum degree of the tree during the sequence of insertions. Central to our algorithm is the construction and maintenance of separator trees of small height. These algorithms may be of independent interest. However, due to lack of space we have omitted the details on separator trees. For further details we refer the reader to the full version of the paper [5].

The rest of this paper is organized as follows. In Sect. 2 we define separator trees and state results on the construction and efficiently maintenance of separator trees of small height. In Sect. 3 we present our algorithm for constructing and maintaining evolutionary trees. In Sect. 4 and 5 the lower bound is proved using an explicit adversary argument. The adversary strategy used is an extension of an adversary used by Borodin, Guibas, Lynch, and Yao [3] for proving

a trade-off between the preprocessing time of a set of elements and membership queries, and Brodal, Chaudhuri, and Radhakrishnan [4] for proving a trade-off between the update time of a set of elements and the time for reporting the minimum of the set.

2 Separator Trees

In this section we define separator trees and state results about efficient algorithms for their constructing and maintenance. For further details see [5].

Definition 1. *Let T be an unrooted tree with n nodes. A* separator tree *S_T for T is a rooted tree on the same set of nodes, defined recursively as follows: The root of S_T is a node u in T, called the* separator node*. The removal of u from T disconnects T into disjoint trees $T_1, \ldots, T_k$, where k is the number of edges incident to u in T. The children of u in S_T are the roots of separator trees for $T_1, \ldots, T_k$.*

Clearly, there are many possible separator trees S_T for a given tree T. An example is shown in Fig. 2.

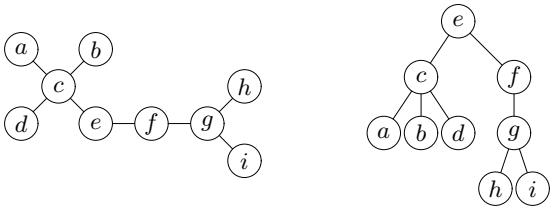


Fig. 2. A tree T (left) and a separator tree S_T for T (right)

For later use, we note the following facts for separator trees:

Fact 1 *Let S_T be a separator tree for T, and let v be a node in T. If S_v denotes the subtree of S_T rooted at v, then:*

1. *The subgraph T_v induced by the nodes in S_v is a tree, and S_v is a separator tree for T_v.*
2. *For any edge from T with exactly one endpoint in T_v, the other endpoint is an ancestor of v in S_T, and each ancestor of v can be the endpoint of at most one such edge.*

The main point of a separator tree S_T is that it may be balanced, even when the underlying tree T is not balanced for any choice of root. The notion of balanced separator trees is contained in the following definition, where the size $|T|$ of a tree T denotes the number of nodes in T, and where T_i refers to the trees $T_1, \ldots, T_k$ from Definition 1.

Definition 2. *A separator tree is a* t-separator tree*, for a threshold* $t \in [1/2, 1]$*, if* $|T_i| \leq t|T|$ *for each* T_i *and the separator tree for each* T_i *is also a* t*-separator tree.*

In [5] we first give a simple algorithm for constructing 1/2-separator trees in time $O(n \log n)$. We then improve the running time of the algorithm to $O(n)$ by adopting additional data structures. We note that a 1/2-separator tree has height at most $\lfloor \log n \rfloor$.

We also consider dynamic separator trees under the insertion of new nodes into a tree T and its corresponding separator tree S_T, and show how to maintain separators trees with small height in logarithmic time per insertion. Our methods for maintaining balance and height in separator trees during insertions of new nodes are based on rebuilding of subtrees, and are inspired by methods of Andersson and Lai described in [1,2] for maintaining small height in binary search trees. We first show how the linear time construction algorithm for 1/2-separator trees leads to a simple algorithm for keeping separator trees well balanced. The height bound achieved by this algorithm is $O(\log n)$, using $O(\log n)$ amortized time per update. We then use a two-layered structure to improve the height bound to $\log n + O(1)$ without sacrificing the time bound. The improved constant factor in the height bound is significant for our use of separator trees for maintaining evolutionary trees in the experiment model, since the number of experiments for an insertion of a new species will turn out to be proportional to the height of the separator tree. Furthermore, this height bound is within an additive constant of the best bound possible, as trees exist where any separator tree must have height at least $\lfloor \log n \rfloor$, e.g. a tree which is a single path.

Finally, we extend the separator trees with a specific ordering of the children, facilitating our use of separator trees in Sect. 3 for finding insertion points for new species in evolutionary trees. The basic idea is to speed up the search in the separator tree by considering the children of the nodes in decreasing size-order. This ensures a larger reduction of subtree size in the case that many children have to be considered before the subtree to proceed the search in is found. Our main result about separator trees is summarized in the following theorem.

Theorem 1. *Let* T *be an unrooted tree initially containing* n *nodes. After* $O(n)$ *time preprocessing, an ordered separator tree for* T *can in time* $O(m \log(n+m))$ *be maintained during* m *insertions in a way such that the height is bounded by* $\log(n+m) + 5$ *and such that for any path* $(v_1, v_2, \ldots, v_\ell)$ *from the root* v_1 *to a node* v_ℓ *in the separator tree, the followings holds*

$$\prod_{d_i \leq 2} 2 \cdot \prod_{d_i > 2} d_i \quad < \quad 16d(n+m) \ , \tag{1}$$

where d_i *is the number which* v_{i+1} *has in the ordering of the children of* v_i*, for* $1 \leq i < \ell$*, and* d *is* $\max\{d_1, \ldots, d_{\ell-1}\}$.

3 Algorithm for Constructing and Maintaining Evolutionary Trees

In this section we describe an algorithm for constructing an evolutionary tree T in the experiment model for a set of n species in time $O(nd \log_d n)$, where d is the degree of the tree. Note that d is not known by the algorithm in advance. The algorithm is a further development of an algorithm by Lingas *et al.* in [8]. Our algorithm also supports the insertion of new species with running time $O(md \log_d(n+m))$ using at most $m\lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1}(n+m) + O(1))$ experiments for $d > 2$, and at most $m(\log(n+m) + O(1))$ experiments for $d = 2$, where n is the number of species in the tree to begin with, m is the number of insertions, and d is the maximum degree of the tree during the sequence of insertions.

The construction algorithm inserts one species at the time into the tree in time $O(d \log_d n)$ until all n species have been inserted. The search for the insertion point of a new species a is guided by a separator tree S_T for the internal nodes of the evolutionary tree T for the species inserted so far. The search starts at the root of S_T. In a manner to be described below, we decide by experiments which subtree, rooted at a child of the root in S_T, the search should continue in. This is repeated recursively until the correct insertion point in T for a is found. We keep links between corresponding nodes in S_T and T for switching between the two trees. To facilitate the experiments, we for each internal node in T maintain a pointer to an arbitrary leaf in its subtree. When inserting a new internal node in T this pointer is set to point to the new leaf which caused the insertion of the node.

We say that the insertion point of a is *incident* to a node v, if

1. a should be inserted directly below v, or
2. a should split an edge which is incident to v by creating a new internal node on the edge and make a a leaf below the new node, or
3. if v is the root of T, a new root of T should be created with a and v as its two children.

The invariant for the search is the following. Assume we have reached node v in the separator tree for the internal nodes in T, and let S_v be the internal nodes of T which are contained in the subtree of S_T rooted at v (including v). Then the insertion point of the new species a is incident to a node in S_v.

Let v be the node in S_T for which we want to decide if the insertion point for the new species a is in the subtree above v in T; if it is in a subtree rooted at a child of v in T; or if a should be inserted as a new child of v. We denote by $u_1, \ldots, u_k$ the children of v in T, where $u_1, \ldots, u_{k'}$ are nodes in distinct subtrees $T_1, \ldots, T_{k'}$ below v in S_T, whereas $u_{k'+1}, \ldots, u_k$ are leaves in T or are nodes above v in S_T. The order of the subtrees $T_1, \ldots, T_{k'}$ below v in S_T is given by the ordered separator tree S_T and determines the order of $u_1, \ldots, u_{k'}$. The remaining children $u_{k'+1}, \ldots, u_k$ of v may appear in any order.

We perform at most $\lceil k/2 \rceil$ experiments at v. The i'th experiment is on the species a, b and c, where b and c are leaves in T below u_{2i-1} and u_{2i} respectively.

The leaves b and c can be located using the pointers stored at u_{2i-1} and u_{2i}. Note that the least common ancestor of b and c in T is v. If k is odd then the species b and c in the $\lceil k/2 \rceil$'th experiment is chosen as leaves in T below u_k and u_1 respectively, and note that the two leaves are distinct because $k \geq 2$ by definition. There are four possible outcomes of the i'th experiment corresponding to Fig. 1:

1. (a, b, c) implies that the insertion point for a is incident to a descendent of u_j, where b and c are not descendents of u_j, or a is a new leaf below v.
2. $((a, b), c)$ implies that the insertion point for a is incident to a descendent of u_{2i-1}, since the least common ancestor of a and b is below v in T.
3. $((a, c), b)$ is symmetric to the above case and the insertion point of a is incident to a descendent of u_{2i} (u_1 for the $\lceil k/2 \rceil$'th experiment if k odd).
4. $((b, c), a)$ implies that the insertion point of a is in the subtree above v, since the least common ancestor of a and b is above v. If v is the present root of T, a new root should be created with children a and v.

We perform experiments for increasing i until we get an outcome difference from Case 1, or until we have performed all $\lceil k/2 \rceil$ experiments all with outcome cf. Case 1. In the latter case species a should be inserted directly below v in T as a new child. In the former case, when the outcome of an experiment is different from Case 1, we know in which subtree adjacent to v in T the insertion point for species a is located. If there is no corresponding subtree below v in S_T, then we have identified the edge incident to v in T which the insertion of species a should split. Otherwise we continue recursively searching for the insertion point for species a at the child of v in S_T which roots the separator tree for the subtree adjacent to v which has been identified to contain the insertion point for a. When the insertion point for species a is found, we insert one leaf and at most one internal node into T, and S_T is updated according to Theorem 1.

Lemma 1. *Given an evolutionary tree T for n species with degree d, and a separator tree S_T for T according to Theorem 1, then a new species a can be inserted into T and S_T in amortized time $O(d \log_d n)$ using at most $\lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1} n + O(1))$ experiments for $d > 2$, and at most $\log n + O(1)$ experiments for $d = 2$.*

Proof. Let $v_1, \ldots, v_\ell$ be the nodes in S_T (and T) visited by the algorithm while inserting species a, where v_1 is the root of S_T and v_{j+1} is a child of v_j in S_T. Define d_i by v_{i+1} being the d_i'th child of v_i in S_T, for $1 \leq i < \ell$.

For $d = 2$ we perform exactly one experiment at each v_i. The total number of experiments is thus bounded by the height of the separator tree. By Theorem 1 it follows that the number of experiments is bounded by $\log n + O(1)$. In the following we consider the case where $d \geq 3$.

For $i < \ell$, let x_i denote the number of experiments performed at node v_i. We have $x_i \leq \lceil d/2 \rceil$ and $d_i \geq 2x_i - 1$, since each experiment considers two children of v_i in T and the first experiment also identifies if a should be inserted into the subtree above v_i. At v_ℓ we perform at most $\lceil d/2 \rceil$ experiments.

For $d_1, \ldots, d_{\ell-1}$ we from Theorem 1 have the constraint $\prod_{d_i \leq 2} 2 \cdot \prod_{d_i > 2} d_i \leq 16dn$, since $|S_T| \leq n-1$. To prove the stated bound on the worst case number of experiments we must maximize $\sum_{i=0}^{\ell} x_i$ under the above constraints. We have

$$\begin{aligned}
\log(16dn) &\geq \sum_{d_i \leq 2} 1 + \sum_{d_i > 2} \log d_i \\
&\geq \sum_{x_i = 1} 1 + \sum_{x_i > 1} \log d_i \\
&\geq \sum_{x_i = 1} x_i + \sum_{x_i > 1} x_i \frac{1}{x_i} \log(2x_i - 1) \\
&\geq \frac{1}{\lceil d/2 \rceil} \log(2\lceil d/2 \rceil - 1) \sum_{i=1}^{\ell-1} x_i \ ,
\end{aligned}$$

where the second inequality holds since $x_i > 1$ implies $d_i \geq 3$. The last inequality holds since for $f(x) = \frac{1}{x} \log(2x-1)$ we have $1 > f(2) > f(3)$ and $f(x)$ is decreasing for $x \geq 3$, i.e. $f(x)$ is minimized when x is maximized.

We conclude that $\sum_{i=1}^{\ell-1} x_i \leq \lceil d/2 \rceil \log_{2\lceil d/2 \rceil - 1}(16dn)$, i.e. for the total number of experiments we have $\sum_{i=1}^{\ell} x_i \leq \lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1}(16dn) + 1)$.

The time needed for the insertion is proportional to the number of experiments performed plus the time to update S_T. By Theorem 1 the total time is thus $O(d \log_d n)$. □

From Lemma 1 and Theorem 1 we get the following bounds for constructing and maintaining an evolutionary tree under the insertion of new species in the experiment model.

Theorem 2. *After $O(n)$ preprocessing time an evolutionary tree T for n species can be maintained under m insertions in time $O(dm \log_d(n+m))$ using at most $m\lceil d/2 \rceil (\log_{2\lceil d/2 \rceil - 1}(n+m) + O(1))$ experiments for $d > 2$, and at most $m(\log(n+m) + O(1))$ experiments for $d = 2$, where d is the maximum degree of the tree during the sequence of insertions.*

4 Adversary for Constructing Evolutionary Trees

To prove a lower bound on the number of experiments required for constructing an evolutionary tree of n species with degree at most d, we describe an adversary strategy for deciding the outcome of experiments. The adversary is required to give consistent answers, i.e. the reported outcome of an experiment is not allowed to contradict the outcome of previously performed experiments. A construction algorithm is able to construct an unambiguous evolutionary tree based on the performed experiments when the adversary is not able to answer any additional experiments in such a way that it contradicts the constructed evolutionary tree. The role of the adversary is to force any construction algorithm

to perform provably many experiments in order to construct an unambiguous evolutionary tree.

To implement the adversary strategy for deciding the outcome of experiments in a consistent way, the adversary maintains a rooted infinite d-ary tree, D, where each of the n species are stored at one of the nodes, allowing nodes to store several species. Initially all n species are stored at the root. For each experiment performed, the adversary can move the species downwards by performing a sequence of *moves*, where each move shifts a species from the node it is currently stored at to a child of the node.

By deciding the outcome of experiments, the adversary reveals information about the evolutionary relationships between the species to the construction algorithm performing the experiments. The distribution of the n species on D represents the information revealed by the adversary (together with the forbidden and conflicting lists introduced below). The evolutionary tree T to be established by the construction algorithm will be a connected subset of nodes of D including the root. Initially, when all species are stored at the root, the construction algorithm has no information about the evolutionary relationships. The evolutionary relationships revealed to the construction algorithm by the current distribution of the species on D corresponds to the tree formed by the paths from the root of D to the nodes storing at least one species. More precisely, the correspondence between the final evolutionary tree T and the current distribution of the species on D is that if v is a leaf of T labeled a then species a is stored at some node on the path in D from the root to the node v.

Our objective is to prove that if an algorithm computes T, then the n species on average must have been moved $\Omega(\log_d n)$ levels down by the adversary, and that the number of moves by the adversary is a fraction $O(1/d)$ of the number of experiments performed. These two facts imply the $\Omega(nd \log_d n)$ lower bound on the number of experiments required.

To control its strategy for moving species on D, the adversary maintains for each species a a *forbidden list* $\mathrm{F}(a)$ of nodes and a *conflicting list* $\mathrm{C}(a)$ of species. If a is stored at node v, then $\mathrm{F}(a)$ is a subset of the children $c_1, \ldots, c_d$ of v, and $\mathrm{C}(a)$ is a subset of the other species stored at v. If $c_i \in \mathrm{F}(a)$, then a is not allowed to be moved to child c_i, and if $b \in \mathrm{C}(a)$ then a and b must be moved to two distinct children of v. It will be an invariant that $b \in \mathrm{C}(a)$ if and only if $a \in \mathrm{C}(b)$. Initially all forbidden and conflicting lists are empty. The adversary maintains the forbidden and conflicting lists such that the size of the forbidden and conflicting lists of a species a is bounded by the invariant

$$|\mathrm{F}(a)| + |\mathrm{C}(a)| \leq d - 2 \ . \tag{2}$$

The adversary uses the sum $|\mathrm{F}(a)|+|\mathrm{C}(a)|$ to decide when to move a species a one level down in D. Whenever the invariant (2) becomes violated because $|\mathrm{F}(a)| + |\mathrm{C}(a)| = d - 1$, for a species a stored at a node v, the adversary moves a to a child $c_i \notin \mathrm{F}(a)$ of v. Since $|\mathrm{F}(a)| \leq d - 1$, such a $c_i \notin \mathrm{F}(a)$ is guaranteed to exist. When moving a from v to c_i, the adversary updates the forbidden and conflicting lists as follows: For all $b \in \mathrm{C}(a)$, a is deleted from $\mathrm{C}(b)$ and c_i is

inserted into $F(b)$. If c_i was already in $F(b)$, the sum $|F(b)|+|C(b)|$ decreases by one, if c_i was not in $F(b)$ the sum remains unchanged. Finally, $F(a)$ and $C(a)$ are assigned the empty set.

For two species a and b, we define their *least common ancestor*, $LCA(a,b)$, to be the least common ancestor of the two nodes storing a and b in D. We denote $LCA(a,b)$ as *fixed* if it cannot be changed by future moves of a and b by the adversary. If $LCA(a,b)$ is fixed then the least common ancestor of the two species a and b in T is the node $LCA(a,b)$. If a is stored at node v_a and b is stored at node v_b, it follows that $LCA(a,b)$ is fixed if and only if one of the following four conditions is satisfied.

1. $v_a = LCA(a,b) = v_b$ and $a \in C(b)$ (and $b \in C(a)$).
2. $v_a \neq LCA(a,b) = v_b$ and $c_i \in F(b)$, where c_i is the child of v_b such that the subtree rooted at c_i contains v_a.
3. $v_a = LCA(a,b) \neq v_b$ and $c_i \in F(a)$, where c_i is the child of v_a such that the subtree rooted at c_i contains v_b.
4. $v_a \neq LCA(a,b) \neq v_b$.

In Case 1, species a and b are stored at the same node and cannot be moved to the same child because $a \in C(b)$, i.e. $LCA(a,b)$ is fixed as the node which currently stores a and b. Cases 2 and 3 are symmetric. In Case 2, species a is stored at a descendant of a child c_i of the node storing b, and b cannot be moved to c_i because $c_i \in F(b)$, i.e. $LCA(a,b)$ is fixed as the node which currently stores b. Finally, in Case 4, species a and b are stored at nodes in disjoint subtrees, i.e. $LCA(a,b)$ is already fixed.

The operation $Fix(a,b)$ ensures that $LCA(a,b)$ is fixed as follows:

1. If $v_a = LCA(a,b) = v_b$ and $a \notin C(b)$ then insert a into $C(b)$ and insert b into $C(a)$.
2. If $v_a \neq LCA(a,b) = v_b$ and $c_i \notin F(b)$, where c_i is the child of v_b such that the subtree rooted at c_i contains v_a, then insert c_i into $F(b)$.
3. If $v_a = LCA(a,b) \neq v_b$ and $c_i \notin F(a)$, where c_i is the child of v_a such that the subtree rooted at c_i contains v_b, then insert c_i into $F(a)$.

Otherwise $Fix(a,b)$ does nothing. If performing $Fix(a,b)$ increases $|F(a)|$ such that $|F(a)|+|C(a)| = d-1$, then a is moved one level down as described above. Similarly, if $|F(b)|+|C(b)| = d-1$ then b is moved one level down. After performing $Fix(a,b)$ we thus have that $|F(a)|+|C(a)| \leq d-2$ and $|F(b)|+|C(b)| \leq d-2$, which ensures that the invariant (2) is not violated.

When the construction algorithm performs an experiment on three species a, b and c, the adversary decides the outcome of the experiment based on the current distribution of the species on D and the content of the conflicting and forbidden lists. To ensure the consistency of future answers, the adversary first fix the least common ancestors of a, b and c by applying the operation Fix three times: $Fix(a,b)$, $Fix(a,c)$ and $Fix(b,c)$. After having fixed $LCA(a,b)$, $LCA(a,c)$, and $LCA(b,c)$, the adversary decides the outcome of the experiment by examining $LCA(a,b)$, $LCA(a,c)$, and $LCA(b,c)$ in D as described below. The four cases correspond to the four possible outcomes of an experiment cf. Fig. 1.

1. If $\mathrm{LCA}(a,b) = \mathrm{LCA}(b,c) = \mathrm{LCA}(a,c)$ then return (a,b,c).
2. If $\mathrm{LCA}(a,b) \neq \mathrm{LCA}(b,c) = \mathrm{LCA}(a,c)$ then return $((a,b),c)$.
3. If $\mathrm{LCA}(a,c) \neq \mathrm{LCA}(a,b) = \mathrm{LCA}(b,c)$ then return $((a,c),b)$.
4. If $\mathrm{LCA}(b,c) \neq \mathrm{LCA}(a,b) = \mathrm{LCA}(a,c)$ then return $((b,c),a)$.

5 Lower Bound Analysis

We will argue that the above adversary strategy forces any construction algorithm to perform at least $\Omega(nd \log_d n)$ experiments before being able to conclude unambiguously the evolutionary relationships between the n species.

Theorem 3. *The construction of an evolutionary tree for n species requires $\Omega(nd \log_d n)$ experiments, where d is the degree of the constructed tree.*

Proof. We first observe that an application of $\mathrm{Fix}(a,b)$ at most increases the size of the two conflicting lists, $\mathrm{C}(a)$ and $\mathrm{C}(b)$, by one, or the size of one of the forbidden list, $\mathrm{F}(a)$ or $\mathrm{F}(b)$, by one. If performing $\mathrm{Fix}(a,b)$ increases the sum $|\mathrm{F}(a)| + |\mathrm{C}(a)|$ to $d-1$, then species a is moved one level down in D and $\mathrm{F}(a)$ and $\mathrm{C}(a)$ are emptied, which causes the overall sum of the sizes of forbidden and conflicting lists to decrease by $d-1$. This implies that a total of k Fix operations, starting with the initial configuration where all conflicting and forbidden lists are empty, can cause at most $2k/(d-1)$ moves. Since an experiment involves three Fix operations, we can bound the total number of moves during m experiments by $6m/(d-1)$.

Now consider the configuration, i.e. the distribution of species and the content of conflicting and forbidden lists, when the construction algorithm computing the evolutionary tree terminates. Some species may have nonempty forbidden lists or conflicting lists. By forcing one additional move on each of these species as described in Sect. 4, we can guarantee that all forbidden and conflicting lists are empty. At most n additional moves must be performed.

Let T' be the tree formed by the paths in D from the root to the nodes storing at least one species. We first argue that all internal nodes of T' have at least two children. If a species has been moved to a child of a node, then the forbidden list or conflicting list of the species was nonempty. If the forbidden list was nonempty, then each of the forbidden subtrees already contained at least one species, and if the conflicting list was nonempty there was at least one species on the same node that was required to be moved to another subtree, at the latest by the n additional moves. It follows that if a species has been moved to a child of a node then at least one species has been moved to another child of the node, implying that T' has no node with only one child.

We next argue that all n species are stored at the leaves of T' and that each leaf of T' stores either one or two species. If there is a non-leaf node in T' that still contains a species, then this species can be moved to at least two children already storing at least one species in the respective subtrees, implying that the adversary can force at least two distinct evolutionary trees which are consistent with the answers returned. This is a contradiction. It follows that all species

are stored at leaves of T'. If a leaf of T' stores three or more species, then an experiment on three of these species can generate different evolutionary trees, which again is a contradiction. We conclude that each leaf of T' stores exactly one or two species, and all internal nodes of T' store no species. It follows that T' has at least $n/2$ leaves.

For a tree with k leaves and degree d, the sum of the depths of the leaves is at least $k \log_d k$. Since each leaf of T' stores at most two species, the n species can be partitioned into two disjoint sets of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$ such that in each set all species are on distinct leaves of T'. The sum of the depths of all species is thus at least $\lceil n/2 \rceil \log_d \lceil n/2 \rceil + \lfloor n/2 \rfloor \log_d \lfloor n/2 \rfloor \geq n \log_d (n/2)$. Since the depth of a species in D is equal to the number of times the species has been moved one level down in D, and since m experiments generate at most $6m/(d-1)$ moves and we perform at most n additional moves, we get the inequality

$$n \log_d (n/2) \;\leq\; 6m/(d-1) + n \;\;,$$

from which the lower bound $m \geq (d-1)n(\log_d(n/2) - 1)/6$ follows. □

References

1. A. Andersson. Improving partial rebuilding by using simple balance criteria. In *Proc. 1st Workshop on Algorithms and Data Structures (WADS)*, volume 382 of *Lecture Notes in Computer Science*, pages 393–402. Springer-Verlag, 1989.
2. A. Andersson and T. W. Lai. Fast updating of well-balanced trees. In *Proc. 2nd Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 447 of *Lecture Notes in Computer Science*, pages 111–121. Springer-Verlag, 1990.
3. A. Borodin, L. J. Guibas, N. A. Lynch, and A. C. Yao. Efficient searching using partial ordering. *Information Processing Letters*, 12:71–75, 1981.
4. G. S. Brodal, S. Chaudhuri, and J. Radhakrishnan. The randomized complexity of maintaining the minimum. *Nordic Journal of Computing, Selected Papers of the 5th Scandinavian Workshop on Algorithm Theory (SWAT)*, 3(4):337–351, 1996.
5. G. S. Brodal, R. Fagerberg, C. N. S. Pedersen, and A. Östlin. The complexity of constructing evolutionary trees using experiments. Technical Report BRICS-RS-01-1, BRICS, Department of Computer Science, University of Aarhus, 2001.
6. S. K. Kannan, E. L. Lawler, and T. J. Warnow. Determining the evolutionary tree using experiments. *Journal of Algorithms*, 21:26–50, 1996.
7. M. Y. Kao, A. Lingas, and A. Östlin. Balanced randomized tree splitting with applications to evolutionary tree constructions. In *Proc. 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 184–196, 1999.
8. A. Lingas, H. Olsson, and A. Östlin. Efficient merging, construction, and maintenance of evolutionary trees. In *Proc. 26th Int. Colloquium on Automata, Languages and Programming (ICALP)*, volume 1644 of *Lecture Notes in Computer Science*, pages 544–553. Springer-Verlag, 1999.
9. C. G. Sibley and J. E. Ahlquist. Phylogeny and classification of birds based on the data of DNA-DNA-hybridization. *Current Ornithology*, 1:245–292, 1983.

Hidden Pattern Statistics

Philippe Flajolet[1], Yves Guivarc'h[2],
Wojciech Szpankowski[3], and Brigitte Vallée[4]

[1] Algorithms Project, INRIA-Rocquencourt, 78153 Le Chesnay, France
[2] IRMAR, Université de Rennes I, F-35042 Rennes Cedex, France
[3] Dept. Computer Science, Purdue University, W. Lafayette, IN 47907, U.S.A
[4] GREYC, Université de Caen, F-14032 Caen Cedex, France

Abstract. We consider the sequence comparison problem, also known as "hidden pattern" problem, where one searches for a given *subsequence* in a text (rather than a string understood as a sequence of consecutive symbols). A characteristic parameter is the number of occurrences of a given pattern w of length m as a subsequence in a random text of length n generated by a memoryless source. Spacings between letters of the pattern may either be constrained or not in order to define valid occurrences. We determine the mean and the variance of the number of occurrences, and establish a Gaussian limit law. These results are obtained via combinatorics on words, formal language techniques, and methods of analytic combinatorics based on generating functions and convergence of moments. The motivation to study this problem comes from an attempt at finding a reliable threshold for intrusion detections, from textual data processing applications, and from molecular biology.

1 Introduction

String matching and *sequence comparison* are two basic problems of pattern matching known informally as "stringology". Hereafter, by a string we mean a sequence of consecutive symbols. In string matching, given a pattern $w = w_1 w_2 \dots w_m$ (of length m) one searches for some/all occurrences of w (as a block of consecutive symbols) in a text T_n of length n. The algorithms by Knuth–Morris–Pratt and Boyer–Moore [7] provide efficient ways of finding such occurrences. Accordingly, the number of string occurrences in a random text has been intensively studied over the last two decades, with significant progress in this area being reported [3,9,10,15,16,17,24]. For instance Guibas and Odlyzko [9, 10] have revealed the fundamental rôle played by autocorrelation vectors and their associated polynomials. Régnier and Szpankowski [16,17] established that the number of occurrences of a string is asymptotically normal under a diversity of models that include Markov chains. Nicodème, Salvy, and Flajolet [15] showed generally that the number of places in a random text at which a 'motif' (i.e., a general regular expression pattern) terminates is asymptotically normally distributed.

In sequence comparisons, we search for a given pattern $\mathcal{W} = w_1 w_2 \dots w_m$ in the text $T_n = t_1 t_2 \dots t_n$ as a *subsequence*, that is, we look for indices $1 \le i_1 <$

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 152–165, 2001.

$i_2 < \cdots < i_m \leq n$ such that $t_{i_1} = w_1$, $t_{i_2} = w_2$, $\cdots$, $t_{i_m} = w_m$. We also say that the word w is *"hidden"* in the text; thus we call this the *hidden pattern* problem. For example, `date` occurs as a subsequence in the text `hidden pattern`, in fact four times, but not even once as a string. We can impose an additional set of constraints $\mathcal{D}$ on the indices $i_1, i_2, \ldots, i_m$ to record a valid subsequence occurrence: for a given family of integers d_j ($d_j \geq 1$, possibly $d_j = \infty$), one should have $(i_{j+1} - i_j) \leq d_j$. In other words, the allowed lengths of the "gaps" $(i_{j+1} - i_j - 1)$ should be $< d_j$. With # representing a 'don't-care-symbol' (similar to the unix '$\star$'-convention) and the subscript denoting a strict upper bound on the length of the associated gap, a typical pattern may look like

$$\texttt{ab}\#_2\texttt{r}\#\texttt{ac}\#\texttt{a}\#\texttt{d}\#_4\texttt{a}\#\texttt{br}\#\texttt{a};$$

there, # abbreviates $\#_\infty$ and $\#_1$ is omitted; the meaning is that '`ab`' should occur first contiguously, followed by '`r`' with a gap of < 2 symbols, followed anywhere later in the text by '`ac`', etc. The case when all the d_j's are infinite is called the *unconstrained problem*; when all the d_j's are finite, we speak of the *constrained* problem. The case where all d_j reduce to 1 gives back classical string matching as a limit case.

Motivations. Our original motivation to study this problem came from *intrusion detection* in the area of computer security. The problem is important due to the rise of attacks on computer systems. There are several approaches to intrusion detections, but, recently the pattern matching approach has found many advocates, most notably in [2,14,25]. The main idea of this approach is to search in an audit file (the text) for certain patterns (known also as signatures) representing suspicious activities that might be indicative of an intrusion by an outsider, or misuse of the system by an insider. The key to this approach is to recognize that these patterns are **subsequences** because an intrusion signature specification requires the possibility of a variable number of intervening events between successive events of the signature. In practice one often needs to put some additional restrictions on the distance between the symbols in the searched subsequence, which leads to the constrained version of subsequence pattern matching. The fundamental question is then: *How many occurrences of a signature (subsequence) constitute a real attack?* In other words, how to set a *threshold* so that we can detect only real intrusions and avoid false alarms? It is clear that *random* (unpredictable) events occur and setting the threshold too low will lead to an unrealistic number of false alarms. On the other hand, setting the threshold too high may result in missing some attacks, which is even more dangerous. This is a fundamental problem that motivated our studies of hidden pattern statistics. By knowing the most likely number of occurrences and the probability of deviating from it, we can set a threshold such that with a small probability we miss real attacks.

Molecular biology provides another important source of applications [18,23,24]. As a rule, there, one searches for subsequences, not strings. Examples are in abundance: split genes where *exons* are interrupted by *introns*, *starting* and *stopping* signal in genes, *tandem repeats* in DNA, etc. In general, for gene searching, the constrained hidden pattern matching (perhaps with an exotic constraint set) is the right approach for finding meaningful information. The hidden pattern

problem can also be viewed as a close relative of the longest common subsequence (LCS) problem, itself of immediate relevance to computational biology and still surrounded by mystery [20].

We, computer scientists and mathematicians, are certainly not the first who invented hidden words and hidden meaning [1]. Rabbi Akiva in the first century A.D. wrote a collection of documents called *Maaseh Merkava* on secret mysticism and meditations. In the eleventh century Spanish Solomon Ibn Gabirol called these secret teachings *Kabbalah*. Kabbalists organized themselves as a secret society dedicated to study of the ancient wisdom of Torah, looking for mysterious connections and hidden truth, meaning, and words in Kaballah and elsewhere (without computers!). Recent versions of this activity are *knowledge discovery and data mining, bibliographic search, lexicographic research, textual data processing*, or even *web site indexing*. Public domain utilities like `agrep`, `grappe`, `webglimpse` (developed by Manber and Wu [26], Kucherov [13], and others) depend crucially on approximate pattern matching algorithms for subsequence detection. Many interesting algorithms, based on regular expressions and automata, dynamic programming, directed acyclic word graphs, digital tries or suffix trees have been developed; see [5,8,13,26] for a flavour of the diversity of approaches.

In all of the contexts mentioned above, it is of obvious interest to discern what constitutes a meaningful observation of pattern occurrences from what is merely a statistically unavoidable phenomenon (noise!). This is precisely the problem addressed here. We establish *hidden pattern statistics*—i.e., precise probabilistic information on number of occurrences of a given pattern w as a subsequence in a random text T_n generated by a memoryless source, this in the most general case (covering the constrained and unconstrained versions as well as mixed situations). Surprisingly enough and to the best of our knowledge, there are no results in the literature that address the question at this level of generality. An immediate consequence of our results is the possibility to set *thresholds* at which appearance of a (subsequence) pattern starts being meaningful.

Results. Let Ω_n be the number of occurrences of a given pattern $\mathcal{W}$ as a subsequence in a random text of length n generated by a memoryless source (i.e., symbols are drawn independently). We investigate the general case where we allow some of the gaps to be restricted, and others to be unbounded. Then the most important parameter is the quantity b defined as the number of unbounded gaps (the number of indices j for which $d_j = \infty$) plus 1; the product D of all the finite constraints d_j plays also a rôle. We obtain the mean, the variance, all moments, and finally a central limit law. Precisely, we prove in Theorem 1 that the number of occurrences has mean and variance given by

$$\mathbf{E}[\Omega_n] \sim \frac{n^b}{b!}\, D\, \pi(\mathcal{W}), \quad \mathbf{Var}[\Omega_n] \sim \sigma^2(\mathcal{W})\, n^{2b-1}$$

where $\pi(\mathcal{W})$ is the probability of $\mathcal{W}$, and $\sigma^2(\mathcal{W})$ is a computable constant that depends explicitly (though intricately) on the structure of the pattern $\mathcal{W}$ and the constraints. Then we prove the central limit law by moment methods, that is, we show that all centered moments $(\Omega_n - \mathbf{E}[\Omega_n])/n^{b-\frac{1}{2}}$ converge to the appropriate

moments of the Gaussian distribution (Theorem 2). We stress that, except in the constrained case, the difficulty of the analysis lies in a nonlinear growth of the mean and the variance so that many standard approaches to establishing the central limit law tend to fail.

For the unconstrained problem, one has $b = m$, and both the mean and the variance admit pleasantly simple closed forms. For the constrained case, one has $b = 1$, while the mean and the variance become of linear growth. To visualize the dependency of $\sigma^2(\mathcal{W})$ of $\mathcal{W}$, we observe that, when all the d_j equal 1, the problem reduces to traditional *string matching* that was extensively studied in the past as witnessed by the (incomplete) list of references: [3,9,10, 15,16,17,24]. It is well known that for string matching the variance coefficient is a function of the so-called *autocorrelation* of the string. In the general case of hidden pattern matching, the autocorrelation must be replaced by a more complex quantity that depends on the way pairs of constrained occurrences may intersect (cf. Theorem 1).

Methodology. The way we approach the probabilistic analysis is through a formal description of situations of interest by means of regular languages. Basically such a description of *contexts* of one, two, or several occurrences gives access to expectation, variance, and higher moments, respectively. A systematic translation into *generating functions* is available by methods of analytic combinatorics deriving from the original Chomsky-Schützenberger theorem. Then, the structure of the implied generating functions at the pole $z = 1$ provides the necessary asymptotic information. In fact, there is an important phenomenon of *asymptotic simplification* where the essentials of combinatorial-probabilistic features are reflected by the singular forms of generating functions. For instance, variance coefficients come out naturally from this approach together with, for each case, a suitable notion of correlation; higher moments are seen to arise from a fundamental singular symmetry of the problem, a fact that eventually carries with it the possibility of estimating moments. From there Gaussian laws eventually result by basic moment convergence theorems. Perhaps the originality of the present approach lies in such a joint use of combinatorial-enumerative techniques and of analytic-probabilistic methods.

2 Framework

We fix an alphabet $\mathcal{A} := \{a_1, a_2, \ldots, a_r\}$. The text is $T_n = t_1 t_2 \cdots t_n$. A particular matching problem is specified by a pair $(\mathcal{W}, \mathcal{D})$: the *pattern* $\mathcal{W} = w_1 \cdots w_m$ is a word of length m; the *constraint* $\mathcal{D} = (d_1, \ldots, d_{m-1})$ is an element of $(\mathbf{N}^+ \cup \{\infty\})^{m-1}$.

Positions and occurrences. An m-tuple $I = (i_1, i_2, \ldots, i_m)$ $(1 \le i_1 < i_2 < \cdots < i_m)$ satisfies the constraint $\mathcal{D}$ if $i_{j+1} - i_j \le d_j$, in which case it is called a *position*. Let $\mathcal{P}_n(\mathcal{D})$ be the set of all positions subject to the separation constraint $\mathcal{D}$, satisfying furthermore $i_m \le n$. An *occurrence* of pattern $\mathcal{W}$ in the text T_n of length n subject to the constraint $\mathcal{D}$ is a position $I = (i_1, i_2, \ldots, i_m)$ of $\mathcal{P}_n(\mathcal{D})$ for which $t_{i_1} = w_1$, $t_{i_2} = w_2, \ldots, t_{i_m} = w_m$. For a text T_n of length n,

the number of occurrences $\Omega_n(\mathcal{D})$ (of w) subject to the constraint $\mathcal{D}$ is then a sum of characteristic variables

$$\Omega_n(\mathcal{D}) = \sum_{I \in \mathcal{P}_n(\mathcal{D})} X_I, \quad \text{with} \quad X_I := [\![w \text{ occurs at position } I \text{ in } T_n]\!], \tag{1}$$

where $[\![B]\!] = 1$ if the property B holds, and $[\![B]\!] = 0$ otherwise (Iverson's notation).

Blocks and aggregates. In the general case, the subset $\mathcal{F}$ of indices j for which d_j is finite ($d_j < \infty$) has cardinality $m - b$ with $1 \le b \le m$. The two extreme values of b, namely, $b = m$ and $b = 1$, thus describe the (fully) unconstrained and the (fully) constrained problem respectively. The subset $\mathcal{U}$ of indices j for which d_j is unbounded ($d_j = \infty$) has cardinality $b - 1$. It separates the pattern $\mathcal{W}$ into b independent subpatterns that are called the blocks and are denoted by $\mathcal{W}_1, \mathcal{W}_2, \ldots \mathcal{W}_b$. All the possible d_j "inside" $\mathcal{W}_r$ are finite and form the subconstraint $\mathcal{D}_r$. In the example described in the introduction, one has $b = 6$ and the six blocks are

$\mathcal{W}_1 = \mathtt{a}\#_1\mathtt{b}\#_2\mathtt{r}$, $\mathcal{W}_2 = \mathtt{a}\#_1\mathtt{c}$, $\mathcal{W}_3 = \mathtt{a}$, $\mathcal{W}_4 = \mathtt{d}\#_4\mathtt{a}$, $\mathcal{W}_5 = \mathtt{b}\#_1\mathtt{r}$, $\mathcal{W}_6 = \mathtt{a}$.

In the same way, an occurrence $I = (i_1, i_2, \ldots, i_m)$ of $\mathcal{W}$ subject to constraint $\mathcal{D}$ gives rise to b subpositions $I^{[1]}, I^{[2]}, \ldots I^{[b]}$, the rth term $I^{[r]}$ being an occurrence of $\mathcal{W}_r$ subject to constraint $\mathcal{D}_r$. The rth *block* $B^{[r]}$ is the closed segment whose end points are the extremal elements of $\mathcal{I}^{[r]}$, and the *aggregate* of position I, denoted by $\alpha(I)$, is the collection of these b blocks. In the example of the introduction, the position

$$I = (6, 7, 9, 18, 19, 22, 30, 33, 50, 51, 60)$$

satisfies the constraint $\mathcal{D}$ and gives rise to six subpositions,

$I^{[1]} = (6, 7, 9)$, $I^{[2]} = (18, 19)$, $I^{[3]} = 22$, $I^{[4]} = (30, 33)$, $I^{[5]} = (50, 51)$, $I^{[1]} = 60$;

accordingly, the resulting aggregate $\alpha(I)$ is formed with six blocks,

$B^{[1]} = [6, 9]$, $B^{[2]} = [18, 19]$, $B^{[3]} = [22]$, $B^{[4]} = [30, 33]$, $B^{[5]} = [50, 51]$, $B^{[6]} = [60]$.

Probabilistic model. We consider a *memoryless source* that emits symbols of the text independently and denote by p_α ($0 < p_\alpha < 1$) the probability of the symbol $\alpha \in \mathcal{A}$ being emitted. For a given length n, a random *text*, denoted by T_n is drawn according to the product probability on $\mathcal{A}^n$. For instance, the pattern probability $\pi(\mathcal{W})$ is defined by $\pi(\mathcal{W}) = \prod_{i=1}^{n} p_{w_i}$, a quantity that surfaces throughout the analysis. Under this randomness model, the quantity $\Omega_n(\mathcal{D})$ becomes a *random variable* that is itself a sum of correlated random variables X_I (defined in (1)) for all allowable $I \in \mathcal{P}_n(\mathcal{D})$.

Generating functions. We shall consider throughout this paper structures superimposed on words. For a class $\mathcal{V}$ of structures and given a weight function c (induced by the probabilities of individual letters), we introduce the *generating function*

$$V(z) \equiv \sum_n V_n z^n := \sum_{v \in \mathcal{V}} c(v) z^{|v|},$$

where the size $|v|$ is the number of letters involved in the structure. Then[1], $V_n = [z^n]V(z)$ is the total weight of all structures of size n. The collection of occurrences is described by means of regular expressions extended with disjoint unions, and Cartesian products. It is then known that disjoint unions and Cartesian products correspond respectively to sums and products of generating functions; see [19,21] for a general framework. Such correspondences make it possible to translate symbolically combinatorial descriptions into generating function equations and a great use is made of this in what follows. All the resulting generating functions turn out to be *rational*, of the form $V(z) = (1-z)^{-(k+1)}P(z)$ for some integer $k \geq 0$ and polynomial P, so that

$$V_n := [z^n]\frac{1}{(1-z)^{k+1}}P(z) = \frac{n^k}{k!}\,P(1)\,\left(1+O(\frac{1}{n})\right). \tag{2}$$

3 Mean and Variance Estimates of the Number of Occurrences

Mean value analysis. The first moment analysis is easily obtained by describing the collection of all occurrences in terms of formal languages. Let $\mathcal{O}$ be the collection of all occurrences of $\mathcal{W}$ as a hidden word. Each occurrence can be viewed as a "context" with an initial string, then the first letter of the pattern, then a separating string, then the second letter, etc. The collection $\mathcal{O}$ is then described by

$$\mathcal{O} = \mathcal{A}^\star \times \{w_1\} \times \mathcal{A}^{<d_1} \times \{w_2\} \times \mathcal{A}^{<d_2} \times \ldots \times \{w_{m-1}\} \times \mathcal{A}^{<d_{m-1}} \times \{w_m\} \times \mathcal{A}^\star. \tag{3}$$

There, for $d < \infty$, $\mathcal{A}^{<d}$ denotes the collection of all words of length strictly less d, i.e., $\mathcal{A}^{<d} := \bigcup_{i<d}\mathcal{A}^i$, whereas, for $d = \infty$, $\mathcal{A}^{<\infty}$ denotes the collection of all finite words, i.e., $\mathcal{A}^{<\infty} := \mathcal{A}^\star = \bigcup_{i<\infty}\mathcal{A}^i$. The associated generating functions are

$$A_d(z) = 1 + z + z^2 + \cdots + z^{d-1} = \frac{1-z^d}{1-z}, \quad A_\infty(z) = 1 + z + z^2 + \cdots = \frac{1}{1-z}.$$

We now weight each occurrence by the quantity $\pi(w) = E[X_I]$, so that the generating function $O(z)$ of $\mathcal{O}$ coincides with the generating function of the expectations $\mathbf{E}[\Omega_n]$,

$$O(z) = \sum_{n\geq 1}\mathbf{E}[\Omega_n]\; z^n = \left(\frac{1}{1-z}\right)^{b+1} \times \left(\prod_{i=1}^{m} p_{w_i} z\right) \times \left(\prod_{i\in\mathcal{F}} \frac{1-z^{d_i}}{1-z}\right), \tag{4}$$

and, with $\pi(\mathcal{W})$ the probability of the pattern $\mathcal{W}$, one finds from (2) and (4):

$$\mathbf{E}[\Omega_n] = [z^n]O(z) = \frac{n^b}{b!}\left(\prod_{i\in\mathcal{F}} d_i\right)\;\pi(\mathcal{W})\left(1+O\left(\frac{1}{n}\right)\right).$$

[1] The notation $[z^n]f(z)$ represents the coefficient of z^n in the series $f(z)$.

Variance analysis. For variance and higher moment analysis, it is essential to work with centred random variables defined as

$$Y_I := X_I - \mathbf{E}[X_I] = X_I - \pi(\mathcal{W}), \quad \Xi_n(\mathcal{D}) := \Omega_n(\mathcal{D}) - \mathbf{E}[\Omega_n(\mathcal{D})] = \sum_{I \in \mathcal{P}_n(\mathcal{D})} Y_I.$$

The second moment of the centred variable $\Xi_n(\mathcal{D})$ equals the variance of $\Omega_n(\mathcal{D})$ and with the centred variables defined above one has

$$\mathbf{E}[\Xi_n^2(\mathcal{D})] = \sum_{I,J \in \mathcal{P}_n(\mathcal{D})} \mathbf{E}[Y_I Y_J].$$

There are two kinds of pairs (I, J) according as they intersect or not. When I and J do not intersect, the corresponding random variables Y_I and Y_J are independent, and the corresponding covariance $E[Y_I Y_J]$ reduces to 0. It is thus sufficient to consider intersecting subsets I and J. Suppose that there exist two occurrences of pattern $\mathcal{W}$ at positions I and J which intersect at ℓ distinct places, the k-th intersection point being the r_k-th in the natural ordering of I and the s_k-th in the natural ordering of J. (This is only possible if, for all $k, 1 \le k \le \ell$, one has $w_{r_k} = w_{s_k}$.) We then denote by $\mathcal{W}_{I \cap J}$ the subpattern of $\mathcal{W}$ that occurs at position $I \cap J$, and by $\pi(\mathcal{W}_{I \cap J})$ the probability of this subpattern. Since the expectation $\mathbf{E}[X_I X_J]$ equals $\pi(\mathcal{W})^2/\pi(\mathcal{W}_{I \cap J})$, the expectation $\mathbf{E}[Y_I Y_J] = \mathbf{E}[X_I X_J] - \pi(w)^2$ involves a correlation number $e(I, J)$

$$\mathbf{E}[Y_I Y_J] = \pi^2(\mathcal{W})\ e(I, J), \quad \text{with} \quad e(I, J) = \frac{1}{\pi(\mathcal{W}_{I \cap J})} - 1. \tag{5}$$

In this case, we take the pair of occurrences relative to (I, J) as weighted by $E[Y_I Y_J]$, and consider the collection $\mathcal{O}_2$ of pairs of intersecting occurrences. The associated generating function $O_2(z)$ coincides with the generating function of the expectations $E[Y_I Y_J]$, that is,

$$O_2(z) = \sum_{n \ge 1} z^n \sum_{\substack{I,J \in \mathcal{P}_n(\mathcal{D}),\\ I \cap J \neq \emptyset}} E[Y_I Y_J] = \sum_{n \ge 1} z^n \mathbf{E}[\Xi_n^2(\mathcal{D})].$$

We now need to estimate $O_2(z)$ as $z \to 1$. First, define the *aggregate* $\alpha(I, J)$ to be the system of blocks obtained by merging together all intersecting blocks of the two aggregates $\alpha(I)$ and $\alpha(J)$. The number of blocks $\beta(I, J)$ of $\alpha(I, J)$ plays a fundamental rôle here, since it measures the *degree of freedom* of pairs. Since I and J intersect, there exists at least one block of $\alpha(I)$ that intersects a block of $\alpha(J)$, so that $\beta(I, J)$ is at most equal to $2b - 1$. Next, we group the sets I, J according to the value of $\beta(I, J)$ and write $\mathcal{O}_2^{[p]}$ for the collection of intersecting pairs (I, J) of occurrences for which $\beta(I, J)$ equals $2b - p$. Since there is a fundamental translation invariance, we introduce a notion of *full pairs*: a pair (I, J) of $\mathcal{P}_q(\mathcal{D}) \times \mathcal{P}_q(\mathcal{D})$ is *full* if the aggregate $\alpha(I, J)$ completely covers the interval $[1, q]$. (Clearly, the possible values of q are finite.) Then the collection $\mathcal{O}_2^{[p]}$ is isomorphic to $(\mathcal{A}^\star)^{2b-p+1} \times \mathcal{B}_2^{[p]}$, where $\mathcal{B}_2^{[p]}$ is the subset of full pairs such that $\beta(I, J)$ equals $2b - p$. The generating function of $\mathcal{O}_2^{[p]}$ is accordingly

$$O_2^{[p]}(z) = \left(\frac{1}{1-z}\right)^{2b-p+1} \times B_2^{[p]}(z).$$

Here, $B_2^{[p]}(z)$ is the generating function of the collection $\mathcal{B}_2^{[p]}$ and from our earlier discussion, it is a *polynomial* of degree at most $2d(m-1)+1$, with $d = \max_{i\in\mathcal{F}} d_i$. Now, an easy dominant pole analysis entails that $[z^n]O_2^{[p]} = O(n^{2b-p})$. This proves that the dominant contribution to the variance is given by $[z^n]O_2^{[1]}$, which is of order $O(n^{2b-1})$. Then, the variance $\mathbf{E}[\Xi_n^2]$ involves the constant $B_2^{[1]}(1)$ that is the total weight of the collection $\mathcal{B}_2^{[1]}$; the polynomial $B_2^{[1]}(z)$ is itself the generating function of the collection $\mathcal{B}_2^{[1]}$, conceptually an extension of Guibas and Odlyzko's autocorrelation polynomial.

Since the standard deviation is of an order, $O(n^{b-1/2})$, that is smaller than the mean, $O(n^b)$, concentration of distribution holds, via a well-known argument based on Chebyshev's inequalities. In summary:

Theorem 1. *Consider a general constraint $\mathcal{D}$ and the number of occurrences $\Omega_n \equiv \Omega_n(\mathcal{D})$. The mean and variance of Ω_n satisfy*

$$\begin{aligned}\mathbf{E}[\Omega_n] &= \frac{\pi(\mathcal{W})}{b!}\left(\prod_{j\in\mathcal{F}} d_j\right) n^b \left(1+O(\frac{1}{n})\right),\\ \mathbf{Var}[\Omega_n] &= \sigma^2(\mathcal{W}) n^{2b-1}\left(1+O(\frac{1}{n})\right),\end{aligned}$$

where $\mathcal{F}$ is the set of j such that $d_j < \infty$, and the "variance coefficient" $\sigma^2(\mathcal{W})$ involves the autocorrelation $\kappa(\mathcal{W})$

$$\sigma^2(\mathcal{W}) = \frac{\pi^2(\mathcal{W})}{(2b-1)!}\,\kappa^2(\mathcal{W}) \quad \textit{with} \quad \kappa^2(\mathcal{W}) := \sum_{(I,J)\in\mathcal{B}_2^{[1]}} \left(\frac{1}{\pi(\mathcal{W}_{I\cap J})} - 1\right). \tag{6}$$

The set $\mathcal{B}_2^{[1]}$ is the collection of all pairs of occurrences (I,J) that satisfy three conditions: (i) they are full; (ii) they are intersecting; (iii) there is a single pair (r,s) with $1 \le r,s \le b$ for which the rth block $B^{[r]}$ of $\alpha(I)$ and the sth block $C^{[s]}$ of $\alpha(J)$ intersect.

Computation of the variance. The computation of the autocorrelation $\kappa(\mathcal{W})$ reduces to b^2 computations of correlations $\kappa(\mathcal{W}_r, \mathcal{W}_s)$, relative to pairs $(\mathcal{W}_r, \mathcal{W}_s)$ of blocks. Note that each correlation of the form $\kappa(\mathcal{W}_r, \mathcal{W}_s)$ involves a totally constrained problem and can be evaluated by dynamic programming. Precisely, one has

$$\kappa^2(\mathcal{W}) = D^2 \sum_{1\le r,s\le b} \frac{1}{D_r D_s}\binom{r+s-2}{r-1}\binom{2b-r-s}{b-r}\,\kappa(\mathcal{W}_r,\mathcal{W}_s), \tag{7}$$

where $\kappa(\mathcal{W}_r, \mathcal{W}_s)$ is the sum of the $e(I,J)$ taken over all full intersecting pairs (I,J) formed with an occurrence I of $\mathcal{W}_r$ subject to constraint $\mathcal{D}_r$ and an occurrence J of $\mathcal{W}_s$ subject to constraint $\mathcal{D}_s$. Let us explain the formula (7) in words: for a pair (I,J) of the set $\mathcal{B}_2^{[1]}$, there is a single pair (r,s) of indices with $1 \le r,s \le b$ for which the rth block $B^{[r]}$ of $\alpha(I)$ and the sth block $C^{[s]}$ of $\alpha(J)$

intersect. Then, there exist $r+s-2$ blocks before the block $\alpha(B^{[r]}, C^{[s]})$ and $2b-r-s$ blocks after it. We then have three different degrees of freedom: (i) the relative order of blocks $B^{[i]}(i<r)$ and blocks $C^{[j]}(j<s)$, and similarly the relative order of blocks $B^{[i]}(i>r)$ and blocks $C^{[j]}(j>s)$; (ii) the lengths of the blocks (there are D_j possible lengths for the jth block); (iii) finally the relative positions of the blocks $B^{[r]}$ and $C^{[s]}$.

In the unconstrained problem, the parameter b equals m, and each block $\mathcal{W}_r$ is reduced to the symbol w_r. Then the " correlation coefficient" $\kappa^2(\mathcal{W})$ simplifies to

$$\kappa^2(\mathcal{W}) := \sum_{1\le r,s\le m} \binom{r+s-2}{r-1}\binom{2m-r-s}{m-r} \Gamma(r,s)\left(\frac{1}{p_{w_r}}-1\right), \qquad (8)$$

where the "autocorrelation matrix" Γ of pattern $\mathcal{W}$ is defined by $\Gamma(r,s) := [\![w_r = w_s]\!]$.

4 Central Limit Laws

Our goal is to prove that Ω_n appropriately normalized tends to the standard normal distribution. We consider the following normalized random variable

$$\widetilde{\Xi}_n := \frac{\Xi_n}{n^{b-1/2}} = \frac{\Omega_n - \mathbf{E}[\Omega_n]}{n^{b-1/2}},$$

where b is the number of blocks of the constraint $\mathcal{D}$. We shall show that $\widetilde{\Xi}_n$ behaves asymptotically as a normal variable with mean 0 and standard deviation σ. By the classical *moment convergence theorem* (Theorem 30.2 of [4]) this is established once all moments of $\widetilde{\Xi}_n$ are known to converge to the appropriate moments of the standard normal distribution. We remind the reader that if G is a standard normal variable (i.e., a Gaussian distributed variable with mean 0 and standard deviation 1), then for any integral $s \ge 0$

$$\mathbf{E}[G^{2s}] = 1\cdot 3\cdots(2s-1), \quad \mathbf{E}[G^{2s+1}] = 0. \qquad (9)$$

We shall accordingly distinguish two cases based on the parity of r, $r=2s$ and $r=2s+1$, and prove that

$$\mathbf{E}[\Xi_n^{2s+1}] = o(n^{(2s+1)(b-1/2)}), \quad \mathbf{E}[\Xi_n^{2s}] \sim \sigma^{2s}\,(1\cdot 3\cdots(2s-1))\,n^{2sb-s}, \qquad (10)$$

which implies Gaussian convergence of $\widetilde{\Xi}_n$.

Theorem 2. *The random variable Ω_n asymptotically follows a Central Limit Law:*

$$\lim_{n\to\infty} \Pr\left\{\frac{\Omega_n - \mathbf{E}[\Omega_n]}{\sqrt{\mathbf{Var}[\Omega_n]}} \le x\right\} = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-t^2/2}\,dt \qquad (11)$$

Proof. The proof below is combinatorial; it basically reduces to grouping and enumerating adequately the various combinations of indices in the sum that expresses $\mathbf{E}[\Xi_n^r]$. Once more, $\mathcal{P}_n(\mathcal{D})$ is formed of all the positions of $[1,n]$ subject

to the constraint $\mathcal{D}$ and $\mathcal{P}(\mathcal{D}) = \bigcup_n \mathcal{P}_n(\mathcal{D})$. Then totally distributing the terms in $\Xi_n^r(\mathcal{D})$ yields

$$\mathbf{E}[\Xi_n^r] = \sum_{(I_1,\ldots,I_r)\in\mathcal{P}_n^r(\mathcal{D})} \mathbf{E}[Y_{I_1}\cdots Y_{I_r}]. \tag{12}$$

An r-tuple of sets $(I_1,\ldots,I_r)$ in $\mathcal{P}^r(\mathcal{D})$ is said to be *friendly* if each I_k intersects at least one other I_ℓ, with $\ell \neq k$ and we let $\mathcal{Q}^{(r)}(\mathcal{D})$ be the set of all friendly collections in $\mathcal{P}^r(\mathcal{D})$. For $\mathcal{P}^r$, $\mathcal{Q}^{(r)}$, and their derivatives below, we add the subscript n each time the situation is particularized to texts of length n. If $(I_1,\ldots,I_r)$ does not lie in $\mathcal{Q}^{(r)}(\mathcal{D})$, then $\mathbf{E}[Y_{I_1}\cdots Y_{I_r}] = 0$, since at least one of the Y_I's is independent of the other factors in the product and the Y_I's have been centred, $\mathbf{E}[Y_I] = 0$. One can thus restrict attention to friendly families and get the basic formula

$$\mathbf{E}[\Xi_n^r] = \sum_{(I_1,\ldots,I_r)\in\mathcal{Q}_n^{(r)}(\mathcal{D})} \mathbf{E}[Y_{I_1}\cdots Y_{I_r}], \tag{13}$$

where the expression involves fewer terms than in (12). From there, we proceed in two stages. First, restrict attention to friendly families that give rise to the dominant contribution and introduce a suitable subfamily $\mathcal{Q}_\star^{(r)} \subset \mathcal{Q}^{(r)}$; in so doing, moments of odd order appear to be negligible. Next, for even order r, the family $\mathcal{Q}_\star^{(r)}$ involves a symmetry and it suffices to consider another smaller subfamily $\mathcal{Q}_{\star\star}^{(r)} \subset \mathcal{Q}_\star^{(r)}$ that corresponds to a "standard" form of occurrence intersection; this last reduction precisely gives rise to the even Gaussian moments.

Odd moments. Given $(I_1,\ldots,I_r) \in \mathcal{Q}^{(r)}$, one defines the aggregate $\alpha(I_1,I_2,\ldots,I_r)$ as the aggregation (in the sense of the variance calculation above) of $\alpha(I_1)\cup\cdots\cup\alpha(I_r)$. Next, the *number of blocks* of $(I_1,\ldots,I_r)$ is the number of blocks of the aggregate $\alpha(I_1,\ldots,I_r)$; if p is the total number of intersecting blocks of the aggregate $\alpha(I_1,\ldots,I_r)$, the aggregate $\alpha(I_1,I_2,\ldots I_r)$ has $rb-p$ blocks. Like previously, we say that the family $(I_1,\ldots,I_r)$ of $\mathcal{Q}_q^{(r)}$ is *full* if the aggregate $\alpha(I_1,I_2,\ldots I_r)$ completely covers the interval $[1,q]$. In this case, the length of the aggregate is at most $rd(m-1)+1$, and the generating function of full families is a polynomial $P_r(z)$ of degree at most $rd(m-1)+1$ with $d = \max_{j\in\mathcal{F}} d_j$. Then, the generating function of families of $\mathcal{Q}^{(r)}$ whose block number equals k is of the form

$$\left(\frac{1}{1-z}\right)^{k+1} \times P_r(z),$$

so that the number of families of $\mathcal{Q}_n^{(r)}$ whose block number equals k is $O(n^k)$. This observation proves that the dominant contribution to (13) arises from friendly families with a maximal block number. It is clear that the minimum number of intersecting blocks of any element of $\mathcal{Q}^{(r)}$ equals equals $\lceil r/2\rceil$, since it coincides exactly with the minimum number of edges of a graph with r vertices which contains no isolated vertex. Then the maximum block number of a friendly

family equals $rb - \lceil r/2 \rceil$. In view of this fact and the remarks above regarding cardinalities, we immediately have

$$\mathbf{E}\left[\Xi_n^{2s+1}\right] = O\left(n^{(2s+1)b-s-1}\right) = o\left(n^{(2s+1)(b-1/2)}\right)$$

which establishes the limit form of odd moments in (10).

Even moments. We are thus left with estimating the even moments. The dominant term is relative to friendly families of $\mathcal{Q}^{(2s)}$ with an intersecting block number equal to s, whose set we denote by $\mathcal{Q}_\star^{(2s)}$. In such a family, each subset I_k intersects one and only one other subset I_ℓ. Furthermore, if the blocks of $\alpha(I_h)$ are denoted by $B_h^{[u]}, 1 \le u \le b$, there exists only one block $B_k^{[u_k]}$ of $\alpha(I_k)$ and only one block $B_\ell^{[u_\ell]}$ that contains the points of $I_k \cap I_\ell$. This defines an involution τ such that $\tau(k) = \ell$ and $\tau(\ell) = k$ for all pairs of indices (ℓ, k) for which I_k and I_ℓ intersect. Furthermore, given the symmetry relation $\mathbf{E}[Y_{I_1} \cdots Y_{I_{2s}}] = \mathbf{E}[Y_{I_{\rho(1)}} \cdots Y_{I_{\rho(2s)}}]$ it suffices to restrict attention to friendly families of $\mathcal{Q}_\star^{(2s)}$ for which the involution τ is the standard one with cycles $(1,2)$, $(3,4)$, etc; for such "standard" families whose set is denoted by $\mathcal{Q}_{\star\star}^{(2s)}$, the pairs that intersect are thus $(I_1, I_2), \ldots, (I_{2s-1}, I_{2s})$. Since the set $\mathcal{K}_{2s}$ of involutions of $2s$ elements has cardinality $K_{2s} = 1 \cdot 3 \cdot 5 \cdots (2s-1)$, the equality

$$\sum_{\mathcal{Q}_{\star n}^{(2s)}} \mathbf{E}[Y_{I_1} \cdots Y_{I_{2s}}] = K_{2s} \sum_{\mathcal{Q}_{\star\star n}^{(2s)}} \mathbf{E}[Y_{I_1} \cdots Y_{I_{2s}}], \tag{14}$$

entails that we can work now solely with standard families.

The class of occurrences relative to standard families is $\mathcal{A}^\star \times (\mathcal{A}^\star)^{2sb-s-1} \times \mathcal{B}_{2s}^{[s]} \times \mathcal{A}^\star$, and involves the collection $\mathcal{B}_{2s}^{[s]}$ of all full friendly $2s$-tuples of occurrences with a number of blocks equal to s. Since $\mathcal{B}_{2s}^{[s]}$ is exactly a shuffle of s copies of $\mathcal{B}_2^{[1]}$ (as introduced in the study of the variance), the associated generating function is

$$\left(\frac{1}{1-z}\right)^{2sb-s+1} (2sb-s)! \left(\frac{B_2^{[1]}(z)}{(2b-1)!}\right)^s,$$

where $B_2^{[1]}(z)$ is the already introduced autocorrelation polynomial. Upon taking coefficients, we obtain the estimate

$$\sum_{\mathcal{Q}_{\star\star n}^{(2s)}} \mathbf{E}[Y_{I_1} \cdots Y_{I_{2s}}] \sim n^{(2b-1)s} \sigma^{2s}. \tag{15}$$

In view of the formulæ (12), (13), (14), and (15) above, this yields the estimate of even moments and leads to the second relation of (10). (Note that the even Gaussian moments eventually come out of the number of involutions, which corresponds to a fundamental symmetry present in the problem.) This completes the proof of Theorem 2.

5 Conclusion

As a test case, we took the full text of Hamlet where all nonalphabetic characters are suppressed. This gives us a (rather unpoetical looking) text that has one long line with 30,316 words and $n = 120,057$ alphabetical characters: "*who s there nay answer me stand and unfold yourself long live the king bernardo he you come most carefully upon your hour [...]*". The pattern is "*The law is Gaussian*" [$w =$ thelawisgaussian] and its mirror image $\widetilde{w}$, corresponding to $m = 16$. Based on the empirical distribution of letter frequencies in the text, we anticipate the pattern to appear $1.330\,10^{48}$ times as a subsequence, while the observed counts are $1.365\,10^{48}$ and $1.388\,10^{48}$, a deviation of less than 4% from what is expected. Similarly, if we bound the separation distance between any two letters by d, analysis predicts that the pattern might start occurring near $d = 10$, while its presence is unlikely for smaller values, $d < 10$. In fact, w starts occurring at $d = 14$ while $\widetilde{w}$ starts at $d = 13$—a deviation of some 30–40% from what the model predicts. Here is a table of observed versus predicted values when d varies:

		$w =$ thelawisgaussian		$\widetilde{w} =$ naissuagsiwaleht	
d	Expected (E)	Occurred (Ω)	Ω/E	Occurred (Ω)	Ω/E
13	9.195E+01	0	0.00	18	0.19
14	2.794E+02	693	2.47	371	1.32
20	5.886E+04	124,499	2.11	41,066	0.69
50	5.482E+10	76,146,232,395	1.38	48,386,404,680	0.88
∞	1.330E+48	1.36554E+48	1.03	1.38807E+48	1.04

This (together with many other experiments) shows a fair fit between the theoretical model and the observed data even though the text chosen is far from being "random".

Extensions. For the constrained case where all the distances are finite, based on finite state models and the de Bruijn graph, it is possible to obtain local limit laws (i.e., a direct estimation of probability densities), a characterization of the speed of convergence to the asymptotic limit (it is $n^{-1/2}$), as well as large deviation estimates (that are exponentially small); see the full paper. For the unconstrained case, the corresponding problems appear to be related to products of random matrices and to the difficult case of random walks on nilpotent Lie groups; see Guivarc'h's paper [11] for context and references. Finally, preliminary investigations indicate that the methods developed here apply to Markovian sources and more generally to all dynamical sources in the sense of Vallée [6,22].

Acknowledgments. We thank M. Atallah (Purdue U.) for introducing us to the intrusion detection problem that motivated this study. This research was supported in part by sponsors of CERIAS at Purdue under contract 1419991431A, by the ALCOM-FT Project (# IST-1999-14186) of the European Union, and by NSF Grant C-CR 9804760.

References

1. A. Aczel, *The Mystery of the Aleph. Mathematics, the Kabbalah, and the Search for Infinity*, Four Walls Eight Windows, New York, 2000.
2. A. Apostolico and M. Atallah, Compact Recognizers of Episode Sequences, Submitted to *Information and Computation*.
3. E. Bender and F. Kochman, The Distribution of Subword Counts is Usually Normal, *European Journal of Combinatorics*, 14, 265-275, 1993.
4. P. Billingsley, *Probability and Measure*, Second Edition, John Wiley & Sons, New York, 1986.
5. L. Boasson, P. Cegielski, I. Guessarian, and Yuri Matiyasevich, Window-Accumulated Subsequence Matching Problem is Linear, In *Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems: PODS 1999*, ACM Press, 327–336, 1999.
6. J. Clément, P. Flajolet, and B. Vallée, Dynamical Sources in Information Theory: A General Analysis of Trie Structures, *Algorithmica*, 29, 307–369, 2001.
7. M. Crochemore and W. Rytter, *Text Algorithms*, Oxford University Press, New York, 1994.
8. G. Das, R. Fleischer, L. G asieniec, D. Gunopulos, and J. Kärkkäinen, Episode Matching, In *Combinatorial Pattern Matching, 8th Annual Symposium*, *Lecture Notes in Computer Science* vol. 1264, 12–27, 1997.
9. L. Guibas and A. M. Odlyzko, Periods in Strings, *J. Combinatorial Theory Ser. A*, 30, 19–43, 1981.
10. L. Guibas and A. M. Odlyzko, String Overlaps, Pattern Matching, and Nontransitive Games, *J. Combinatorial Theory Ser. A*, 30, 183–208, 1981.
11. Y. Guivarc'h, Marches aléatoires sur les groupes, *Fascicule de probabilités*, Publ. Inst. Rech. Math. Rennes, 2000.
12. D. E. Knuth, *The Art of Computer Programming, Fundamental Algorithms*, Vol. 1, Third Edition, Addison-Wesley, Reading, MA, 1997.
13. G. Kucherov and M. Rusinowitch, Matching a Set of Strings with Variable Length Don't Cares, *Theoretical Computer Science* 178, 129–154, 1997.
14. S. Kumar and E.H. Spafford, A Pattern-Matching Model for Intrusion Detection, *Proceedings of the National Computer Security Conference*, 11–21, 1994.
15. P. Nicodème, B. Salvy, and P. Flajolet, Motif Statistics, *European Symposium on Algorithms*, *Lecture Notes in Computer Science*, No. 1643, 194–211, 1999.
16. M. Régnier and W. Szpankowski, On the Approximate Pattern Occurrences in a Text, *Proc. Compression and Complexity of SEQUENCE'97*, IEEE Computer Society, 253–264, Positano, 1997.
17. M. Régnier and W. Szpankowski, On Pattern Frequency Occurrences in a Markovian Sequence, *Algorithmica*, 22, 631-649, 1998.
18. I. Rigoutsos, A. Floratos, L. Parida, Y. Gao and D. Platt, The Emergence of Pattern Discovery Techniques in Computational Biology, *Metabolic Engineering*, 2, 159-177, 2000.
19. R. Sedgewick and P. Flajolet, *An Introduction to the Analysis of Algorithms*, Addison-Wesley, Reading, MA, 1995.
20. J. M. Steele, *Probability Theory and Combinatorial Optimization*, SIAM, Philadelphia, 1997.
21. W. Szpankowski, *Average Case Analysis of Algorithms on Sequences*, John Wiley & Sons, New York, 2001.

22. B. Vallée, Dynamical Sources in Information Theory: Fundamental Intervals and Word Prefixes, *Algorithmica*, 29, 262–306, 2001.
23. A. Vanet, L. Marsan, and M.-F. Sagot, Promoter sequences and algorithmical methods for identifying them, *Res. Microbiol.*, 150, 779-799, 1999.
24. M. Waterman, *Introduction to Computational Biology*, Chapman and Hall, London, 1995.
25. A. Wespi, H. Debar, M. Dacier, and M. Nassehi, Fixed vs. Variable-Length Patterns For Detecting Suspicious Process Behavior, *J. Computer Security*, 8, 159-181, 2000.
26. S. Wu and U. Manber, Fast Text Searching Allowing Errors, *Comm. ACM*, 35:10, 83–991, 1995.

Combinatorics and Algorithms on Low-Discrepancy Roundings of a Real Sequence

Kunihiko Sadakane, Nadia Takki-Chebihi, and Takeshi Tokuyama[1]

Graduate School of Information Sciences, Tohoku University,
tokuyama@dais.is.tohoku.ac.jp

Abstract. In this paper, we discuss the problem of computing all the integral sequences obtained by rounding an input real valued sequence such that the discrepancy between the input sequence and each output integral sequence is less than one. We show that the number of such roundings is $n + 1$ if we consider the discrepancy with respect to the set of all subintervals, and give an efficient algorithm to report all of them. Then, we give an optimal method to construct a compact graph to represent the set of global roundings satisfying a weaker discrepancy condition.

1 Introduction

For a given real number α, its *rounding* is either $\lfloor \alpha \rfloor$ or $\lceil \alpha \rceil$. Given a sequence $\boldsymbol{a} = (a_i)_{1 \le i \le n}$ of real numbers, its rounding is an integral sequence $\boldsymbol{b} = (b_i)_{1 \le i \le n}$ such that each entry b_i is a rounding of a_i. Without loss of generality, we can assume that each entry of $\boldsymbol{a}$ is in the closed interval $[0, 1]$. Thus, the rounding of $\boldsymbol{a}$ becomes a binary array.

There are 2^n possible roundings of a given $\boldsymbol{a}$, and we would like to compute good-quality roundings with respect to a given criterion. The problem is not only combinatorially interesting but also related to coding theory, data compression, computer vision, operations research, and Monte Carlo simulation.

In order to give a criterion to determine quality of roundings, we define a distance in the space $\mathcal{A}$ of all $[0, 1]$-valued sequences of n real numbers. For an element $\boldsymbol{a} \in \mathcal{A}$, let $\boldsymbol{a}(I) = \sum_{i \in I} a_i$ be the sum of entries of $\boldsymbol{a}$ whose indices are located in an interval $I \subset [1, n]$. We fix a family of $\mathcal{F}$ of intervals. The l_∞ distance between two elements $\boldsymbol{a}$ and $\boldsymbol{a}'$ in $\mathcal{A}$ with respect to $\mathcal{F}$ is defined by

$$Dist_\infty^{\mathcal{F}}(\boldsymbol{a}, \boldsymbol{a}') = \max_{I \in \mathcal{F}} |\boldsymbol{a}(I) - \boldsymbol{a}'(I)|.$$

$Dist_\infty^{\mathcal{F}}(\boldsymbol{a}, \boldsymbol{b})$ is the rounding error of a rounding $\boldsymbol{b}$ of a given $[0, 1]$-valued sequence $\boldsymbol{a}$ measured by using the distance. The supremum of the optimal rounding error $\sup_{\boldsymbol{a} \in \mathcal{A}} \min_{\boldsymbol{b} \in \mathcal{B}} Dist_\infty^{\mathcal{F}}(\boldsymbol{a}, \boldsymbol{b})$ is called the *inhomogeneous discrepancy* of $\mathcal{A}$ with respect to the family $\mathcal{F}$ [3]. Here, $\mathcal{B}$ is the set of all binary valued sequences of length n. The most popular case is where $\mathcal{F}$ is the set $\mathcal{I}_n$ of all integral subintervals of $[1, n]$, and the discrepancy of with respect to $\mathcal{I}_n$ is sometimes called the *1-dimentional discrepancy* in the literature.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 166–177, 2001.

Abusing the notation, we often call the error $Dist_\infty^{\mathcal{F}}(\boldsymbol{a}, \boldsymbol{b})$ the *discrepancy between* $\boldsymbol{a}$ *and* $\boldsymbol{b}$ with respect to $\mathcal{F}$.

We say that a rounding $\boldsymbol{b}$ of $\boldsymbol{a}$ is an *$\mathcal{F}$-global rounding* if $Dist_\infty^{\mathcal{F}}(\boldsymbol{a}, \boldsymbol{b}) < 1$ holds; in other words, $\boldsymbol{b}$ is a global rounding of $\boldsymbol{a}$ if and only if $\boldsymbol{b}[I]$ is a rounding of $\boldsymbol{a}[I]$ for every $I \in \mathcal{F}$. It is known that for any $\mathcal{F}$, an *$\mathcal{F}$-global rounding* exists. On the other hand, for any constant $\epsilon > 0$, there exists an input $\boldsymbol{a}$ which has no rounding with a discrepancy less than $1 - \epsilon$ even if we consider the family of all intervals of length 2 [1].

There are two classical algorithms each of which computes an $\mathcal{F}$-global rounding (the output sequence depends on the algorithm): One is the error-diffusion algorithm, and the other is Viterbi's decoding algorithm (outlines are given in the appendix). Moreover, Asano et al. [1] have recently shown that for any given input sequence $\boldsymbol{a}$, a binary sequence $\boldsymbol{b}$ minimizing the discrepancy can be computed in time $O(\sqrt{n}|\mathcal{F}|\log^2 n)$, where $|\mathcal{F}|$ is the cardinality of $\mathcal{F}$, and hence $O(n^2)$.

A major defect of the above algorithms is that each of them outputs only one particular $\mathcal{F}$-global rounding. This lack of flexibility causes some serious problems in some applications such as image processing. Therefore, it is desired to design efficient algorithms to output either (1) all $\mathcal{F}$-global roundings or (2) a system so that one can efficiently select a given number of $\mathcal{F}$-global roundings uniformly random from the set of all global roundings.

In this paper, we consider the family $\mathcal{I}_k$ consisting of all intervals of length at most k in $[1, n]$. The family is natural and important in several applications. We first consider the special case where $k = n$, and show that we can report all $\mathcal{I}_n$-global roundings in $O(n^2)$ time. This implies that the number of $\mathcal{I}_n$ global roundings is bounded by a polynomial; indeed, it is at most $n + 1$, and exactly $n+1$ under a non-degeneracy condition. Next, we give an $O(nk)$ time algorithm to output an acyclic network with $O(nk)$ nodes so that the set of all $\mathcal{I}_k$-global roundings equals the set of all directed s-t paths in the network. As byproducts, we show that several optimization rounding problems that can be solved in $O(2^k qn)$ time by using Viterbi's dynamic programming algorithm can be solved in $O(kqn)$ if we restrict the solution space to the set of $\mathcal{I}_k$ global roundings. Here, q is the time to do some basic operations depending on problems. This includes an improved $O(nk)$ time complexity of computing the rounding $\boldsymbol{b}$ minimizing $Dist_\infty^{\mathcal{I}_k}(\boldsymbol{a}, \boldsymbol{b})$.

The present paper mainly focuses on theoretical aspect of the problem; however, our motivation comes from digital halftoning, which is one of the most fundamental techniques in image processing. An intensity image can be considered as a $[0, 1]$-valued $n \times n$ array $\boldsymbol{A}$ where each entry $a_{i,j}$ corresponds to a brightness level (gray level) of the (i, j) pixel of the pixel grid. For a color image, we consider an overlay of three such matrices representing red, green, and blue color components, respectively. The digital halftoning is to compute a binary $n \times n$ array $\boldsymbol{B}$ "approximating" $\boldsymbol{A}$. The intention of this method is to convert a given image which consists of several bits for brightness levels into a binary image having only black and white pixels. This kind of technique is

indispensable to print an image on an output device that produces black dots only, such as facsimiles and laser printers. The problem is not easy; for example, neither simple rounding nor randomized rounding (round each entry $a_{i,j}$ to 1 with probability $a_{i,j}$) generates a good halftoning image.

Up to now, a large number of methods and algorithms for digital halftoning have been proposed (see, e.g., [8,4,9,10]). The ordered dither method [10] and the two-dimensional error diffusion method [4] are quite popular methods. By the nature of the problem, we need help of human's decision to judge the quality of halftoning; however, a nice mathematical measurement for automatically evaluating the quality is desired. Discrepancy is a nice mathemetical measurement for the halftoning [11]. However, two dimensional rounding problem minimizing the discrepancy is NP-hard, and even its approximation is theoretically difficult [1, 2].

The concept and algorithms for global roundings given in this paper will be useful tools for designing nice halftoning methods. Every $\mathcal{I}_k$-global rounding (for a suitable k) gives a good quality rounding for each row. However, if we further consider the side-effect, it is not wise to round each row independently and combine them, since it often causes some systematic patterns (that do not exist in the input image) in the output image: Such a pattern is called a *regular pattern* created by a rounding.

We can avoid generating regular patterns if we have many candidate global roundings for each row and select a suitable one considering the relation to the neighbor rows. Even a random choice of a global rounding works well in our preliminary experiments: Compared to the randomized rounding, the method to choose a global rounding randomly in each row decreases the randomness, and hence tends to keep features of the original image better. Moreover, we can consider several bicriteria optimization problems to compute global a rounding of each row that simultaneously minimizes two-dimensional side effects.

2 Structure of the Set of Global Roundings

2.1 Preliminaries

Let $S(\boldsymbol{a}, \mathcal{F})$ be the set of all $\mathcal{F}$-global roundings of $\boldsymbol{a}$, and let $N(\boldsymbol{a}, \mathcal{F}) = |S(\boldsymbol{a}, \mathcal{F})|$ be the number of different roundings. The discrepancy satisfies the monotonicity by definition; i.e., $Dist^{\mathcal{F}}_{\infty}(\boldsymbol{a}, \boldsymbol{b}) \geq Dist^{\mathcal{J}}_{\infty}(\boldsymbol{a}, \boldsymbol{b})$ if $\mathcal{F} \supset \mathcal{J}$. Therefore, $S(\boldsymbol{a}, \mathcal{F}) \subset S(\boldsymbol{a}, \mathcal{J})$ if $\mathcal{F} \supset \mathcal{J}$.

For a sequence $\boldsymbol{c}$ of length n, let $\boldsymbol{c}(\leq k)$ be its prefix of length k. Thus, $\boldsymbol{a}(\leq k)$ is the prefix of the input sequence $\boldsymbol{a}$ of length k. Abusing the notation, we say that a binary sequence of length k is a $\mathcal{F}$-global rounding of a prefix of $\boldsymbol{a}$ if it is a global rounding of $\boldsymbol{a}(\leq k)$ with respect to $\mathcal{F}(\leq k) = \{I \cap [1, k] : I \in \mathcal{F}\}$. The following lemma is trivial, but useful:

Lemma 1. *The prefix of length k of a $\mathcal{F}$-global rounding $\boldsymbol{b}$ of $\boldsymbol{a}$ is a $\mathcal{F}$-global rounding of the prefix $\boldsymbol{a}(\leq k)$ of $\boldsymbol{a}$. Moreover, for every $\mathcal{F}$-global rounding $\boldsymbol{c}$ of a prefix $\boldsymbol{a}(\leq k)$, its prefix of length $\ell < k$ is a $\mathcal{F}$-global rounding of the prefix $\boldsymbol{a}(\leq \ell)$.*

Definition 1. *A family $\mathcal{F}$ is called prefix-complete if for any $m \leq n$ and for any $I \in \mathcal{F}$, $I \cap [1, m] \in \mathcal{F}$.*

We mainly consider prefix-complete families in this paper. Obviously, $\mathcal{I}_k$, which we focus on, is a prefix-complete family.

2.2 Rounding Graph

Definition 2. *A* rounding graph *of $\boldsymbol{a}$ with respect to $\mathcal{F}$ is a directed acyclic graph G with a source node such that each edge contains either* 0 *or* 1 *as a label, every path from its source to a sink gives a global rounding (if we read the labels at edges on the path sequentially) of $\boldsymbol{a}$, and every global rounding appears as such a path.*

There may be several different rounding graphs for a set of global roundings. We first consider one particular rounding graph (indeed, a binary tree) of an input sequence $\boldsymbol{a}$ with respect to a prefix-complete family $\mathcal{F}$ of intervals. The graph is often called the *keyword tree* in the literature [6], if we consider the set of global roundings as a set of binary keywords. See Figure 1 for an example.

The construction is as follows: We denote $\boldsymbol{b} \bullet 0$ and $\boldsymbol{b} \bullet 1$ as the sequence obtained by appending 0 and 1 to the end of $\boldsymbol{b}$, respectively. We consider a node $v(\boldsymbol{c})$ associated with an integral sequence $\boldsymbol{c}$, and let $V(\boldsymbol{a}, \mathcal{F}) = \{v(\boldsymbol{c}) : \boldsymbol{c}$ is a $\mathcal{F}$-global rounding of a prefix of $\boldsymbol{a}\}$. Here, we use a convention that $\emptyset$ is a global rounding of the empty "prefix" of length 0 of $\boldsymbol{a}$. Consider a graph $\tilde{T}(\boldsymbol{a}, \mathcal{F})$, which has $V(\boldsymbol{a}, \mathcal{F})$ as its node set, and has an arc from $v(\boldsymbol{c})$ to $v(\boldsymbol{d})$ if and only if either $\boldsymbol{d} = \boldsymbol{c} \bullet 0$ or $\boldsymbol{d} = \boldsymbol{c} \bullet 1$: the arc has 0 (resp. 1) as its label in the former (resp. latter) case. The following lemma is immediately obtained from the construction and the definition of a prefix-complete family:

Lemma 2. *$\tilde{T}(\boldsymbol{a}, \mathcal{F})$ is a binary directed tree rooted at $v(\emptyset)$ such that if we read the labels at edges on the path from $v(\emptyset)$ to a node $v(\boldsymbol{c})$ sequentially, we have the binary string $\boldsymbol{c}$.*

The depth of the tree $\tilde{T}(\boldsymbol{a}, \mathcal{F})$ is n by the construction, and we ignore the leaves at shallower levels, if any. In precise, let $T(\boldsymbol{a}, \mathcal{F})$ be the induced subgraph of $\tilde{T}(\boldsymbol{a}, \mathcal{F})$ consisting of nodes on the paths from leaves of level n towards the root. $T(\boldsymbol{a}, \mathcal{F})$ is a rounding graph, since the set of paths from the root to leaves of depth n is exactly the set of $\mathcal{F}$-global roundings. Note that the size of the tree may be exponential in general.

3 $\mathcal{I}_n$-Global Roundings

3.1 Combinatorial Results

We consider the case where $\mathcal{F} = \mathcal{I}_n$. If $N(\boldsymbol{a}, \mathcal{F})$ is very large (say, exponential in n), we have no hope to report all the $\mathcal{F}$-global roundings in polynomial time. The following lemma is easy to prove, but it was a surprising discovery for the authors:

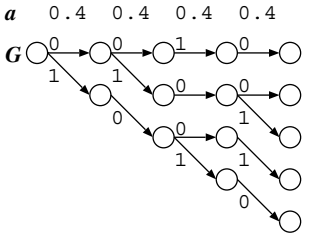


Fig. 1. The rounding graph $T(\boldsymbol{a}, \mathcal{F})$, where $F = \mathcal{I}_n$ and $\boldsymbol{a} = (0.4, 0.4, 0.4, 0.4)$.

Lemma 3. *For any real sequence* $\boldsymbol{a}$ *of length* n, $N(\boldsymbol{a}, \mathcal{I}_n) \leq n + 1$.

Proof. We prove the lemma by induction on n. If $n = 1$, the lemma is trivial. Suppose that the statement holds for each sequence of length less than n. For each rounding $\boldsymbol{b} \in S(\boldsymbol{a}, \mathcal{I}_n)$, we can observe that $\boldsymbol{b}(\leq n-1) \in S(\boldsymbol{a}(\leq n-1), \mathcal{I}_{n-1})$. A pair of binary sequences $\boldsymbol{b}$ and $\boldsymbol{b}'$ is called a *prefix-sharing pair* if $\boldsymbol{b}(\leq n-1) = \boldsymbol{b}'(\leq n-1)$. We claim that there is at most one prefix-sharing pair in $S(\boldsymbol{a}, \mathcal{I}_n)$.

Assume that the claim is false. Thus, we have $\boldsymbol{b}, \boldsymbol{b}', \boldsymbol{c}, \boldsymbol{c}' \in S(\boldsymbol{a}, \mathcal{I}_n)$ such that $\boldsymbol{b}(\leq n-1) = \boldsymbol{b}'(\leq n-1)$, $\boldsymbol{c}(\leq n-1) = \boldsymbol{c}'(\leq n-1)$, and $\boldsymbol{b}(\leq n-1) \neq \boldsymbol{c}(\leq n-1)$. We can assume that the last entries of $\boldsymbol{b}$ and $\boldsymbol{c}$ are 1, while those of $\boldsymbol{b}'$ and $\boldsymbol{c}'$ are 0 entries. Since $\boldsymbol{b}(\leq n-1) \neq \boldsymbol{c}(\leq n-1)$, there exists an interval $[j, n-1]$ such that $\boldsymbol{b}([j, n-1]) \neq \boldsymbol{c}([j, n-1])$. From the definition of the global rounding, $|\boldsymbol{b}([j, n-1]) - \boldsymbol{c}([j, n-1])| = 1$ and without loss of generality, we can assume that $\boldsymbol{b}([j, n-1]) = \boldsymbol{c}([j, n-1]) + 1$. Thus, $\boldsymbol{b}([j, n]) - \boldsymbol{c}'([j, n)] = 2$; however, because of the definition of a global rounding, $\boldsymbol{b}([j, n]) < \boldsymbol{a}([j, n]) + 1$ and $\boldsymbol{c}'([j, n]) > \boldsymbol{a}([j, n]) - 1$ and hence $\boldsymbol{b}([j, n]) < \boldsymbol{c}'([j, n]) + 2$. This gives a contradiction.

From this claim, we have $N(\boldsymbol{a}, \mathcal{I}_n) \leq N(\boldsymbol{a}(\leq n-1), \mathcal{I}_{n-1}) + 1 \leq \{(n-1) + 1\} + 1 = n + 1$, and the lemma is proved. □

Definition 3. *A real sequence* $\boldsymbol{a}$ *is called non-degenerate if* $a(I)$ *is non-integral for every interval* $I \in \mathcal{I}_n$.

Lemma 4. $T(\boldsymbol{a}, \mathcal{I}_n) = \tilde{T}(\boldsymbol{a}, \mathcal{I}_n)$, *and if the sequence* $\boldsymbol{a}$ *is non-degenerate,* $N(\boldsymbol{a}, \mathcal{I}_n) = n + 1$.

Proof. First, we show that for any $k \leq n-1$ and any sequence $\boldsymbol{b} \in S(\boldsymbol{a}(\leq k), \mathcal{I}_k)$, either $\boldsymbol{b} \bullet 0$ or $\boldsymbol{b} \bullet 1$ is a member of $S(\boldsymbol{a}(\leq k+1), \mathcal{I}_{k+1})$. This implies that there is no leaf in $\tilde{T}(\boldsymbol{a}, \mathcal{I}_n)$ in a level with depth $k \leq n-1$, and hence $T(\boldsymbol{a}, \mathcal{I}_n) = \tilde{T}(\boldsymbol{a}, \mathcal{I}_n)$.

Assume that there exists $\boldsymbol{b} \in S(\boldsymbol{a}(\leq k), \mathcal{I}_k)$ such that neither $\boldsymbol{b} \bullet 0$ nor $\boldsymbol{b} \bullet 1$ is a member of $S(\boldsymbol{a}(\leq k+1), \mathcal{I}_{k+1})$. Thus, there exist indices $i \leq k$ and $j \leq k$ such that $\boldsymbol{b}([i, k]) + 1 \geq a([i, k+1]) + 1$ and $\boldsymbol{b}([j, k]) \leq a([j, k+1]) - 1$. Therefore, if $i < j$, we have $\boldsymbol{b}([i, j-1]) \geq 1 + \boldsymbol{a}([i, j-1])$, and otherwise, we have

$\boldsymbol{b}([j, i-1]) \leq -1 + \boldsymbol{a}([j, i-1])$. This is a contradiction, since $|\boldsymbol{a}(I) - \boldsymbol{b}(I)| < 1$ for every interval I.

Next, we show that if $\boldsymbol{a}$ is non-degenerate, there always exists $\boldsymbol{b} \in S(\boldsymbol{a}(\leq k), \mathcal{I}_k)$ such that both of $\boldsymbol{b} \bullet 0$ and $\boldsymbol{b} \bullet 1$ are members of $S(\boldsymbol{a}(\leq k+1), \mathcal{I}_{k+1})$. For the purpose, we use a variant of the error diffusion algorithm (see Appendix) processed in a reverse order starting from k to compute a sequence $\boldsymbol{b}$ such that $0 \geq \boldsymbol{b}([j,k]) - \boldsymbol{a}([j,k+1]) > -1$ for every $j = k-1, k-2, \ldots, 2, 1$. It is not difficult to see that there always exists such a sequence. Because of our assumption that $a(I)$ is not integral, this implies that both of $\boldsymbol{b} \bullet 0$ and $\boldsymbol{b} \bullet 1$ are in $S(\boldsymbol{a}(\leq k+1), \mathcal{I}_{k+1})$. Thus, $|S(\boldsymbol{a}(\leq k+1), \mathcal{I}_{k+1})| \geq |S(\boldsymbol{a}(\leq k), \mathcal{I}_k)| + 1$, and we have $N(\boldsymbol{a}, \mathcal{I}_n) \geq n+1$. Combined with the previous lemma, the inequality must be an equality. □

These two lemmas imply that, if we apply a symbolic perturbation method to modify the input sequence $\boldsymbol{a}$ such that $\boldsymbol{a}(I)$ is non-integral for every I, we can always have exactly $n+1$ global roundings of $\boldsymbol{a}$ with respect to $\mathcal{I}_n$.

One natural question is whether we can obtain a polynomial bound of the number of binary sequences if we relax the discrepancy bound. The answer is negative: suppose that we consider the relaxed condition $Dist_\infty^{\mathcal{I}_n}(\boldsymbol{a}, \boldsymbol{b}) \leq 1$, instead of $Dist_\infty^{\mathcal{I}_n}(\boldsymbol{a}, \boldsymbol{b}) < 1$. Consider the input sequence $\boldsymbol{a}$ of even length whose every entry is 0.5. Then, we can observe that every binary sequences satisfying that $b_{2i-1} + b_{2i} = 1$ for $i = 1, 2, \ldots n/2$ are included in the solution set. There are $2^{n/2}$ such sequences.

3.2 Algorithm for Reporting all $\mathcal{I}_n$-Global Roundings

For the family $\mathcal{I}_n$ of all intervals, we compute all $n+1$ sequences. We indeed construct the rounding graph $T = T(\boldsymbol{a}, \mathcal{I}_n)$ in $O(n^2)$ time and $O(n)$ working space (ignoring the space to store the tree). The tree T is a binary tree of height n with at most $n+1$ leaves, and it has $\Theta(n^2)$ nodes.

For simplicity, we simply call a global rounding for an $\mathcal{I}_n$-global rounding in this subsection. For each global rounding $\boldsymbol{c}$ of a prefix (say, $\boldsymbol{a}(\leq i)$) of $\boldsymbol{a}$, let $\mathit{diff}(\boldsymbol{c}) = \boldsymbol{a}([1,i]) - \boldsymbol{c}([1,i])$. We define $\mathit{maxdiff}(\boldsymbol{c}) = \max\{\mathit{diff}(\boldsymbol{d}) : \boldsymbol{d}$ is a prefix of $\boldsymbol{c}\}$ and $\mathit{mindiff}(\boldsymbol{c}) = \min\{\mathit{diff}(\boldsymbol{d}) : \boldsymbol{d}$ is a prefix of $\boldsymbol{c}\}$.

Starting from $\emptyset$, we construct the tree from top to bottom, increasing the depth one by one. The level which is under construction in the algorithm is called the *current level*. If the current level has a depth i, we construct nodes corresponding to global roundings of $\boldsymbol{a}(\leq i)$. We compute $\mathit{diff}(\boldsymbol{c})$, $\mathit{maxdiff}(\boldsymbol{c})$, and $\mathit{mindiff}(\boldsymbol{c})$ for the nodes in the current level of the tree by using the information of the previous level. Note that $\mathit{maxdiff}(\boldsymbol{c}) < \mathit{mindiff}(\boldsymbol{c}) + 2$ holds.

Suppose that the current level is at depth i, and let $v(\boldsymbol{c})$ be a node of T with depth $i-1$ (the level with depth $i-1$ has been already constructed). We want to decide whether $\boldsymbol{c} \bullet 0$ and/or $\boldsymbol{c} \bullet 1$ are global roundings of $\boldsymbol{a}(\leq i)$. The following result is obtained in a routine way from the definition of a global rounding:

Lemma 5. *Let $\tilde{\boldsymbol{c}}$ be either $\boldsymbol{c} \bullet 0$ or $\boldsymbol{c} \bullet 1$. The sequence $\tilde{\boldsymbol{c}}$ is a global rounding of $\boldsymbol{a}(\leq i)$ if and only if $\mathit{maxdiff}(\boldsymbol{c}) - 1 < \mathit{diff}(\tilde{\boldsymbol{c}}) < \mathit{mindiff}(\boldsymbol{c}) + 1$.*

Since $diff(\boldsymbol{c} \bullet 0) = diff(\boldsymbol{c}) - \boldsymbol{a}(i)$ and $diff(\boldsymbol{c} \bullet 1) = diff(\boldsymbol{c}) + 1 - \boldsymbol{a}(i)$, they can be computed in $O(1)$ time. Thus, we can decide in $O(1)$ time whether $\tilde{\boldsymbol{c}}$ is a global rounding or not. it is easy to see that $maxdiff(\tilde{\boldsymbol{c}})$ and $mindiff(\tilde{\boldsymbol{c}})$ can be computed in $O(1)$ time. Hence, we spend $O(1)$ time to creating a node in the graph T. Thus, the time complexity of our algorithm is $O(n^2)$. Since we only use the information stored in the $(i-1)$-th revel to compute the i-th level, we use $O(n)$ working space (ignoring the space to store the output).

3.3 Compact Rounding Graph for a Smaller Family of Intervals

In some applications, we do not care very long intervals. Hence, instead of $\mathcal{I}_n$, we would like to consider $\mathcal{I}_k$ for $k < n$. Unfortunately, the number of $\mathcal{I}_k$-global roundings is $\Omega((k+1)^{\lfloor n/2k \rfloor})$, and hence exponential in $n/2k$. Therefore, it is too expensive to report all the $\mathcal{I}_k$-global roundings explicitly. Instead, we construct a rounding graph of size $O(nk)$, so that we can generate global roundings in a uniformly random fashion.

Let us learn from the following simple example: Consider a fixed input $\boldsymbol{a} = (0.4, 0.4, \ldots, 0.4)$ consisting of n entries with a value 0.4. A binary string is an $\mathcal{I}_2$-global rounding of $\boldsymbol{a}$ if and only if it contains no two consecutive entries $1, 1$. Such binary sequences correspond to vertices of Fibonacci cube [7], and the number of such sequences equals the $(n+2)$-th Fibonacci number; Hence it is exponential. However, we have a compact rounding graph with $2n+1$ nodes illustrated in the left drawing of the Figure 2. If we consider $\mathcal{I}_3$, we have a rounding graph in the right drawing.

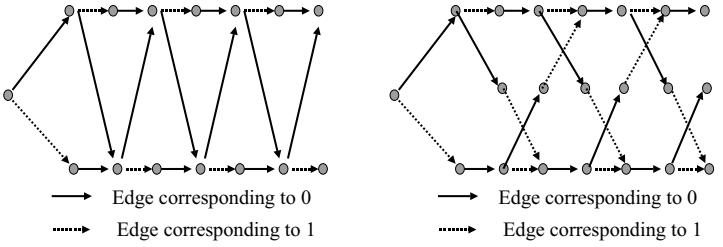


Fig. 2. Rounding graphs for $\mathcal{I}_2$ (left drawing) and $\mathcal{I}_3$ (right drawing).

Theorem 1. *For any input sequence $\boldsymbol{a}$, we can construct its rounding graph with at most $nk + 1 - [k(k+1)/2]$ nodes representing the set of all $\mathcal{I}_k$-global roundings.*

The rest of this subsection is devoted to the proof of the above theorem. The proof is constructive, and similar to the construction of a BDD (bounded decision diagram) from a decision tree. First, we consider the tree $T = \tilde{T}(\boldsymbol{a}, \mathcal{I}_k)$ defined

in the previous section. We say two sequences c and c' are $(k-1)$-similar to each other if they have the same length $\ell \geq k-1$, and they have the same suffix of length $k-1$. The equivalence class of a sequence $\boldsymbol{c}$ under the $(k-1)$-similarlity is denoted by $class(\boldsymbol{c})$. In this subsection, we concentrate on the family $\mathcal{I}_k$, and hence simply write "global roundings" for $\mathcal{I}_k$-global roundings.

Two nodes $v(c)$ and $v(c')$ in T are called similar to each other if c and c' are $(k-1)$-similar. The following claim is easy to verify:

Claim A: If v and v' in T are similar, there is an one-to-one matching between the set of descendants of v and that of v' such that each matching nodes are similar to each other.

We fold the tree T to obtain a graph $G(\boldsymbol{a}, \mathcal{I}_k)$ such that similar nodes are identified and unified into a single node of $G(\boldsymbol{a}, \mathcal{I}_k)$. The edges of T is also unified without causing conflict because of Claim A. Inherited fom T, The graph $G(\boldsymbol{a}, \mathcal{I}_k)$ is a layered directed acyclic graph with $n+1$ layers. From the definition of similarity, the unified edges should have the same label. Due to Claim A, all the outgoing edge with a same label must be unified; thus, each node has at most two outgoing edges. Also, each edge has a label 0 or 1 inherited from T without causing any conflict.

From Lemma 3.1, there are at most k different binary sequences which is a global rounding of a subsequence $a_i, a_{i+1}, ..., a_{i+k-2}$ with respect to $\mathcal{I}_{k-1}$. Hence, at each layer of T, there are at most k different suffixes of the sequences associated to node in the layer. Hence, there are at most k nodes in each layer of G. We can also easily see that the first i-th layer has at most $i+1$ nodes for $i \leq k-1$. This proves the theorem.

3.4 Algorithm to Compute a Compact Rounding Graph

We want to compute $G(\boldsymbol{a}, \mathcal{I}_k)$ efficiently. Since, $\mathcal{I}_k$ is prefix complete, we can apply a similar sweeping strategy to the case of $\mathcal{I}_n$.

Each node of $G(\boldsymbol{a}, \mathcal{I}_k)$ corresponds to an equivalence class of a prefix of $\boldsymbol{a}$, and wrote as $v(\boldsymbol{c})$, where $\boldsymbol{c}$ is the representative of the equivalence class, which is the lexicographically smallest member (in other words, the smallest member if we regard binary sequences as integers in binary forms) in the class.

Starting from $\emptyset$, we construct $G(\boldsymbol{a}, \mathcal{I}_k)$ from the source to sinks, increasing the level (i.e., depth) one by one. If the current level has depth i, we construct vertices corresponding equivalence classes of the global roundings of $\boldsymbol{a}(\leq i)$. As we have shown in the previous subsection, there are at most k such equivalence classes. We maintain $diff(\boldsymbol{c})$, $maxdiff_k(\boldsymbol{c})$, and $mindiff_k(\boldsymbol{c})$ for the representative $\boldsymbol{c}$ of the equivalence class corresponding to each node in the current level of the graph by using the information of the previous level. Let $L(m)$ be the set of representatives of the equivalence classes corresponding to nodes of the m-th level of $G(\boldsymbol{a}, \mathcal{I}_k)$.

Let $\ell(\boldsymbol{c})$ be the length of a sequence $\boldsymbol{c}$. We define $maxdiff_k(\boldsymbol{c}) = \max\{ diff(\boldsymbol{d}) : \boldsymbol{d}$ is a prefix of $\boldsymbol{c}$ such that $\ell(\boldsymbol{d}) \geq \ell(\boldsymbol{c}) - k + 1 \}$ and $mindiff_k(\boldsymbol{c}) = \min\{ diff(\boldsymbol{d}) : \boldsymbol{d}$ is a prefix of $\boldsymbol{c}$ such that $\ell(\boldsymbol{d}) \geq \ell(\boldsymbol{c}) - k + 1 \}$.

Lemma 6. *If* $\boldsymbol{c} = (c_1, c_2, \ldots, c_m)$ *is a prefix of a global rounding with respect to* $\mathcal{I}_k$, $\boldsymbol{c} \bullet c_{m+1}$ *(*$c_{m+1} = 1$ *or* 0*) is a prefix of a global rounding if and only if* $maxdiff_{k-1}(\boldsymbol{c}) + a_{m+1} - 1 < c_{m+1} < mindiff_{k-1}(\boldsymbol{c}) + a_{m+1} + 1$

Hence, we can select all the global roundings among $\{\boldsymbol{c} \bullet 0 : \boldsymbol{c} \in L(m)\}$ and $\{\boldsymbol{c} \bullet 1 : \boldsymbol{c} \in L(m)\}$ in $O(k)$ time. Thus, we can construct $G(\boldsymbol{a}, \mathcal{I}_k)$ in $O(nk + nq)$ time if the following operations can be done in $O(q)$ amortized time for each level: (1): Classify the set of global roundings among $\{\boldsymbol{c} \bullet 0 : \boldsymbol{c} \in L(m)\} \cup \{\boldsymbol{c} \bullet 1 : \boldsymbol{c} \in L(m)\}$ into equivalence classes, and choose representatives. (2): Compute information of *diff*, $mindiff_k$ and $maxdiff_k$ for all representatives in $L(m+1)$.

In order to implement the operation (1), we consider a tree $T(m)$ from the set of representatives $\boldsymbol{c}$ in $L(m)$. The tree has a leaf $l(\boldsymbol{c})$ for each $\boldsymbol{c} \in L(m)$, and each edge has either 0 or 1 as its label, and the path from the root to $l(\boldsymbol{c})$ gives the suffix of length $k-1$ of $\boldsymbol{c}$ in the reverse order. For example, if $k = 4$ and $\boldsymbol{c} = 0, 0, 1, 1, 0, 1, 1$, the path from the root gives the sequence $1, 1, 0$. It is clear that $T(m)$ has $O(k^2)$ edges. From $T(m)$, we can construct $T(m+1)$ by making two copies of $T(m)$, joining them at a new root with edges of labels 0 and 1 respectively, remove leaves which do not correspond to global roundings, and upgrades each other leaf to its parent's place. If two leaves are upgraded to the same position (i.e., if they have the same parent), we know that these two leaves are corresponding to sequences with a same equivalence class.

In order to attain the $O(k)$ time complexity, we use a compressed form $H(m)$ of $T(m)$. Since $T(m)$ has only k leaves, it has at most $k-1$ branching nodes. The vertex set of $H(m)$ consists of the root, leaves, and branching points of $T(m)$. We unite each path between consecutive branching points in $T(m)$ to have an edge of $H(m)$. A label sequence associated with a path in $T(m)$ associated with an edge in $H(m)$ is stored into a cell with $O(k)$ space. Each edge of $H(m)$ has a pointer to the cell containing the label sequence associated with the path in $T(m)$. Instead of updating $T(m)$, we update $H(m)$ into $H(m+1)$. The copying and modifying the structure of $H(m)$ into $H(m+1)$ can be done in $O(k)$ time. We create two cells associated with edges adjacent to the new root. Only at most $O(k)$ cells storing label sequences are updated, and an update of the label sequences is either removing the last bit of the sequence, or appending sequences in two cells; Hence, each such operation can be done in $O(1)$ time. Thus, we can do the operation (1) in $O(k)$ time.

The operation (2) can be implemented in $O(k \log k)$ time by using a dynamic tree data structure. Instead, we do it in $O(k)$ amortized time without using a complicated data structure. We say a level m a major-event level if m is a multiple of k. Other levels are called minor-event levels. At each major-event level, we construct the history of the past k levels used in the following minor-event levels. In precise, consider a major-event level where $m = jk$. For the representative $\boldsymbol{c}$ of each node in the current level, we consider its prefixes $\boldsymbol{c}(\leq i)$ for $(j-1)k < i \leq jk$, and compute $intmin(\boldsymbol{c}[s, jk]) = \min_{s \leq i \leq jk} diff(\boldsymbol{c}(\leq i))$ and $intmax(\boldsymbol{c}[s, jk]) = \max_{s \leq i \leq jk} diff(\boldsymbol{c}(\leq i))$ for each $(j-1)k < s \leq jk$. This computation can be done from right to left in $O(k)$ time for each $\boldsymbol{c}$, and hence $O(k^2)$ time for each major-event level.

At a minor-event level $L(m)$, if jk is the previous major-event level, we compute $localmin(\boldsymbol{c}) = \min_{jk<i\leq m} diff(\boldsymbol{c}(\leq i))$ and $localmax(\boldsymbol{c}) = \max_{jk<i\leq m} diff(\boldsymbol{c}(\leq i))$. Since $localmin(\boldsymbol{c}) = \min\{localmin(\boldsymbol{c}(\leq m-1)), diff(\boldsymbol{c})\}$ (analogous formula holds for $localmax$), they can be computed in $O(1)$ time for each $\boldsymbol{c}$. We can observe that $mindiff_k(\boldsymbol{c}) = \min\{localmin(\boldsymbol{c}), intmin(\boldsymbol{c}[m-k+1, jk])\}$, and we can use the same $intmin$ value for the ancestor of $\boldsymbol{c}$ at the previous major-event level. Analogous formula holds for $maxdiff_k(\boldsymbol{c})$. Hence, the computation at a minor level is $O(k)$. Thus, the amortized time complexity per a level is $O(k)$. Hence, we have obtained the following theorem:

Theorem 2. *The graph $G(\boldsymbol{a}, \mathcal{I}_k)$ can be constructed in $O(nk)$ time using $O(k^2)$ working space.*

We can compute for every node $v(\boldsymbol{c})$ of $G(\boldsymbol{a}, \mathcal{I}_k)$ the number $n(v(\boldsymbol{c}))$ of global roundings of $\boldsymbol{a}$ that have $\boldsymbol{c}$ as their prefix. This can be done in $O(nk)$ time by using a dynamic programming procedure. By using this information, we can generate global roundings uniformly random by walking on the directed acyclic graph $G(\boldsymbol{a}, \mathcal{I}_k)$ (directed from the source to sinks) using $n(v(\boldsymbol{c}))$ as the probability for choosing the next branch (i.e., next bit of the rounding).

4 Fast Viterbi-Type Algorithms and Bicriteria Optimization

Let us review the Viterbi's algorithm (see Appendix) in a general form. For each integral subinterval $J = [i+1, i+k] \subset [1, n]$ of length k, let us consider a function f_J assigning a real value $f_J(\boldsymbol{a}, \boldsymbol{x})$ for each pair of a real sequence $\boldsymbol{a} \in [0,1]^n$ and a binary sequence $\boldsymbol{x} \in \{0,1\}^n$ of length n. The function f_J is called *local* if $f_J(\boldsymbol{a}, \boldsymbol{x})$ is determined by the entries of $\boldsymbol{a}$ and $\boldsymbol{x}$ located in the interval J.

Consider a commutative semigroup operation $\oplus$ satisfying the monotonicity, i.e., if $x_1 \geq y_1$ and $x_2 \geq y_2$ then $x_1 \oplus x_2 \geq y_1 \oplus y_2$. Examples of such operations are $+$, max, min, and taking the L_p norm $(|x_1|^p + |x_2|^p)^{1/p}$. Let us consider the sum (under the $\oplus$ operation) $F(\boldsymbol{a}, \boldsymbol{x}) = \bigoplus_{i=0}^{n-k} f_{[i+1,i+k]}(\boldsymbol{a}, \boldsymbol{x})$, and would like to find a binary sequence $\boldsymbol{x}$ minimizing $F(\boldsymbol{a}, \boldsymbol{x})$.

Viterbi's dynamic programming algorithm can be applied to the above problem. It is easy to see the following: Suppose that $f_J(\boldsymbol{a}, \boldsymbol{x})$ is local and computable in $O(q)$ amortized time if we run the dynamic programming. Then, the binary sequence x minimizing $F(\boldsymbol{a}, \boldsymbol{x})$ can be computed in $O(2^k nq)$ time. If we further combine our global rounding condition, we have the following:

Theorem 3. *Under the assumption as above, the global rounding sequence x of $\boldsymbol{a}$ with respect to $\mathcal{I}_k$ minimizing $F(\boldsymbol{a}, \boldsymbol{x})$ can be computed in $O(knq)$ time.*

Proof. We need to keep $k+1$ binary sequences instead of 2^k sequences in the dynamic programming, because $G(\boldsymbol{a}, \mathcal{I}_k)$ has at most $k+1$ nodes in a level. □

Corollary 1. *The rounding minimizing the L_∞ rounding error with respect to $\mathcal{I}_k$ can be computed in $O(kn)$ time.*

Proof. We set $f_{[i+1,i+k]}(\boldsymbol{a}, \boldsymbol{x})$ to be the maximum of the absolute difference between $\boldsymbol{a}([i+s, i+k])$ and $\boldsymbol{x}([i+s, i+k])$ over $s = 1, 2, \ldots, k$. It is easy to see that $f_J(\boldsymbol{a}, \boldsymbol{x})$ can be computed in $O(1)$ amortized time by using data structures given in previous sections. □

For a family of interval $\mathcal{F}$, we can consider a nonnegative valued function w on $\mathcal{F}$ and define the weighted l_p distance $Dist_p^{\mathcal{F},w}(\boldsymbol{a}, \boldsymbol{b}) = (\sum_{I \in \mathcal{F}} |\boldsymbol{a}(I) - \boldsymbol{b}(I)|^p w(I))^{1/p}$ between $\boldsymbol{a}$ and its rounding $\boldsymbol{b}$. Although a weighted l_p distance is a nice measure of quality of a rounding if we choose suitable w and p, it is time consuming to compute the optimal rounding with respect to this measure [2]. However, if we restrict the solution space to the set of global roundings with respect to $\mathcal{I}_k$, we have the following:

Corollary 2. *Given any weight function w, the global rounding minimizing the weighted l_p error with respect to $\mathcal{I}_k$ can be computed in $O(k^2 n)$ time.*

5 Remarks on Digital Halftoning Applications

From the viewpoint of practical applications, our main target is digital halftoning: We would like to approximate a $[0, 1]$-valued matrix $\boldsymbol{A}$ with a binary matrix $\boldsymbol{B}$. One natural formulation is that we define $Dist_\infty^{\mathcal{F}}(\boldsymbol{A}, \boldsymbol{B}) = \max_{R \in \mathcal{F}} |\boldsymbol{A}(R) - \boldsymbol{B}(R)|$ for a family F of subarrays, and find $\boldsymbol{B}$ minimizing this distance. However, this problem is NP-hard, and even an approximation algorithm with a provable constant approximation ratio is difficult to design [1]. One heuristics method is to round rows one by one, considering the relations to roundings of previous rows. Here, we must keep the rounding of the current row to be similar to the input sequence (the global rounding property certifies it) to reduce the side-effect of roundings of forthcoming rows, and also minimize the two-dimensional error effect in the part of the matrix rounded so far (together with the current row). For the purpose, the bicriteria method given in the preceding section will be suitable. Our experimental results will be reported elsewhere.

References

1. T. Asano, T. Matsui, and T. Tokuyama: "On the complexity of the optimal rounding problems of sequences and matrices," *Proceedings of SWAT00, LNCS1851* (2000), pp. 476-489.
2. T. Asano et al, "Digital Halftoning: Formulation as a combinatorial optimization problem and approximation algorithms based on network flow", working paper, 2000 November.
3. J. Beck and V. T. Sös, *Discrepancy Theory*, in *Handbook of Combinatorics* Volume II, (ed. R.Graham, M. Grötschel, and L Lovász) 1995, Elsevier.
4. R. W. Floyd and L. Steinberg: "An adaptive algorithm for spatial gray scale," *SID 75 Digest, Society for Information Display* (1975), pp. 36–37.

5. H. N. Gabow and R. E. Tarjan: "Faster scaling algorithms for network problems," *SIAM J. Comp.,* 18 (1989), pp. 1013–1036.
6. D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer science and computational biology,* Cambridge U.P. 1997.
7. W. J. Hsu, "Fibonacci cubes – a new interconnection topology," IEEE Trans. Parallel and Distributed Systems, 4 (1993) pp.2–12.
8. D. E. Knuth: "Digital halftones by dot diffusion," ACM Trans. Graphics, 6-4 (1987), pp. 245–273.
9. J. O. Limb: "Design of dither waveforms for quantized visual signals," *Bell Syst. Tech. J.,* 48-7 (1969), pp. 2555–2582.
10. B. Lippel and M. Kurland: "The effect of dither on luminance quantization of pictures," *IEEE Trans. Commun. Tech.,* COM-19 (1971), pp.879-888.
11. V. Rödl and P. Winkler: "Concerning a matrix approximation problem", *Crux Mathmaticorum*, 1990, pp. 76–79.

Appendix: Algorithms for Computing a Global Rounding

Error diffusion algorithm. Let $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ be our input sequence such that $0 \leq a_j \leq 1$ for all $j \in \{1, 2, \ldots, n\}$. The *error diffusion algorithm* computes a binary sequence $\boldsymbol{b}$ from b_1 to b_n greedily in an incremental fashion in linear time. We always keep the difference $\delta_j = \sum_{i=1}^{j}(a_i - b_i)$ if we have already computed b_1 through b_j, and determine b_{j+1} to be 1 if $\delta_j + a_{j+1} > 0.5$ and to be 0 otherwise. It can be easily seen that $-0.5 < \delta_j \leq 0.5$ always holds, and hence for any interval I, $|\sum_{i \in I}(a_i - b_i)| < 0.5 - (-0.5) = 1$.

Viterbi's decoding algorithm is a dynamic programming algorithm that computes a rounding $\boldsymbol{b}$ of a sequence $\boldsymbol{a}$ minimizing $Dist_{\infty}^{\mathcal{F}}(\boldsymbol{a}, \boldsymbol{b})$ for a given $\mathcal{F} \subset \mathcal{I}_k$.

For each binary pattern P of length k, the algorithm computes real numbers $m_0(P, i)$ and $m(P, i)$ for $i = k, k+1, \ldots, n$. The number $m_0(P, i)$ is the discrepancy (with respect to $\mathcal{F}$) between P and the subsequence of $\boldsymbol{a}$ consisting of entries from a_{i-k+1} to a_i.

We compute $m(P, i)$ for all patterns P and all $k \leq i \leq n$ by a dynamic programming procedure: As initialization, we consider the first k entries of $\boldsymbol{a}$, and set $m(P, k) = m_0(P, k)$. Then, we sweep the sequence from left to right to update the rounding error by
$m(P, i) = \max\{m_0(P, i), \min\{m(P^+, i-1), m(P^-, i-1)\}\}$. Here, P^+ and P^- are the patterns by removing the last (i.e., rightmost) bit and appending 1 and 0 to the left of P, respectively. It can be seen that $\min_P\{m(P, n)\}$ attains the minimum of $Dist_{\infty}^{\mathcal{F}}(\boldsymbol{a}, \boldsymbol{b})$ over all binary sequences, and the sequence $\boldsymbol{b}$ can be computed by backtracking the dynamic programming process. The time complexity of this algoirthm is $O(2^k(|\mathcal{F}| + n))$.

We remark that the original Viterbi's decoding algorithm deals with the L_1 measure instead of the L_∞ measure.

Network type algorithm. Asano et al.[1] applied the negative cycle detection algorithm [5] on a network to devise a polynomial time algorithm to compute the rounding sequence $\boldsymbol{b}$ minimizing $Dist_{\infty}^{\mathcal{F}}(\boldsymbol{a}, \boldsymbol{b})$ in $O(\min\{k^2 n \log n, n^{2.5} \log^2 n\})$ time, where k is the maximum length of the intervals of $\mathcal{F}$.

All-Pairs Shortest Paths Computation in the BSP Model

Alexandre Tiskin

No Institute Given

Abstract. The model of bulk-synchronous parallel (BSP) computation is an emerging paradigm of general-purpose parallel computing. We propose a new p-processor BSP algorithm for the all-pairs shortest paths problem in a weighted directed dense graph. In contrast with the general algebraic path algorithm, which performs $O(p^{1/2})$ to $O(p^{2/3})$ global synchronisation steps, our new algorithm only requires $O(\log p)$ synchronisation steps.

1 Introduction

The model of *bulk-synchronous parallel (BSP) computation* (see [18,11,13]) provides a simple and practical framework for general-purpose parallel computing. Its main goal is to support the creation of architecture-independent and scalable parallel software. Key features of BSP are its treatment of the communication medium as an abstract fully connected network, and strict separation of all interaction between processors into point-to-point asynchronous data communication and barrier synchronisation. This separation allows an explicit and independent cost analysis of local computation, communication and synchronisation.

In this paper we propose a new BSP algorithm for the all-pairs shortest paths problem in a weighted directed dense graph. This problem is a special case of the general algebraic path problem, therefore it is natural to compare the general algebraic path algorithm with our new all-pairs shortest paths algorithm. Similarly to the general algorithm, the new algorithm is efficient in local computation, and exhibits a tradeoff between communication and synchronisation; however, our algorithm requires significantly fewer global synchronisation steps.

2 The BSP Model

A *BSP computer*, introduced in [18], consists of p *processors* connected by a communication network. Each processor has a fast *local memory*. The processors may follow different threads of computation. A BSP computation is a sequence of *supersteps*. A superstep consists of an *input phase*, a *local computation phase* and an *output phase*. In the input phase, a processor receives data that were sent to it in the previous superstep; in the output phase, it can send data to other processors, to be received in the next superstep. The processors are synchronised between supersteps. The computation within a superstep is asynchronous.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 178–189, 2001.

Let *cost unit* be the cost of performing a basic arithmetic operation or a local memory access. If, for a particular superstep, w is the maximum number of local operations performed by each processor, h' (respectively, h'') is the maximum number of data units received (respectively, sent) by each processor, and $h = h' + h''$ (another possible definition is $h = \max(h', h'')$), then the cost of the superstep is defined as $w + h \cdot g + l$. Here g and l are the BSP parameters of the computer. The value g is the *communication throughput ratio* (also called "bandwidth inefficiency" or "gap"), the value l is the *communication latency* (also called "synchronisation periodicity"). If a computation consists of S supersteps with costs $w_s + h_s \cdot g + l$, $1 \le s \le S$, then its total cost is $W + H \cdot g + S \cdot l$, where $W = \sum_{s=1}^{S} w_s$ is the local computation cost, $H = \sum_{s=1}^{S} h_s$ is the communication cost, and S is the synchronisation cost. The values of W, H and S typically depend on the number of processors p and on the problem size.

Papers [11,13] present the McColl–Valiant BSP algorithm for standard (non-Strassen) matrix multiplication. The local computation, communication and synchronisation costs of this algorithm are

$$W = O(n^3/p) \qquad H = O(n^2/p^{2/3}) \qquad S = O(1)$$

Paper [12] extends this result to fast (Strassen-type) matrix multiplication. The local computation, communication and synchronisation costs of the extended algorithm are

$$W = O(n^\omega/p) \qquad H = O(n^2/p^{2/\omega}) \qquad S = O(1)$$

where ω is the exponent of fast matrix multiplication (currently 2.376 by [4]).

Many BSP algorithms are only defined for input sizes that are sufficiently large with respect to the number of processors. This requirement is loosely referred to as *slackness*. The algorithm presented is this paper needs a very moderate amount of slackness: to compute all-pairs shortest paths in an n-node graph, we must have $n \ge p$.

For the sake of simplicity, we ignore small irregularities that arise from imperfect matching of integer parameters. For example, when we write "divide an array of size n equally across p processors", the value n may not be an exact multiple of p, and therefore the shares may differ in size by ± 1. We use square bracket notation for matrices, referring to an element of an $n \times n$ matrix A as $A[i, j]$, $1 \le i, j \le n$.

3 Algebraic Path Computation

In this section we consider the problem of finding the closure of a square matrix over a semiring. This problem is also known as the *algebraic path problem*. It unifies many seemingly unrelated computational problems, such as graph connectivity, network reliability, regular language generation, network capacity. All

these tasks can be viewed as instances of the algebraic path problem for an appropriately chosen semiring. More information on applications of the algebraic path problem can be found in [3,19,8,9,14].

Let an $n \times n$ matrix A over a semiring represent a *weighted graph* with nodes $1, \ldots, n$. The *length* of an edge $i \to j$ is defined as the semiring element $A[i,j]$. If the graph is not complete, we assume that non-edges have length zero. We denote semiring addition and multiplication by $\oplus$ and $\odot$ respectively. When it does not create confusion, we also denote semiring multiplication by juxtaposition (e.g. ab for $a \odot b$), and use standard notation for semiring powers (e.g. a^2 for $a \odot a$).

Let $A^* = I \oplus A \oplus A^2 \oplus \cdots$ be the closure of matrix A (it is not guaranteed to exist in a general semiring). The *distance* between nodes i, j is defined as the semiring element $A^*[i,j]$. Note that in this general setting, the distance does not have to correspond to any particular "shortest" path in the graph. In the special case where the semiring is the set of all nonnegative real numbers with ∞, and the operations min and $+$ are used as $\oplus$ and $\odot$ respectively, lengths and distances have their standard graph-theoretic meaning — in particular, ∞ plays the role of the semiring zero, and the distances are realised by shortest paths. We will return to this special case in Section 4.

In order to compute the closure of a square matrix over a general semiring, we use Gaussian elimination without pivoting. In the absence of pivoting, Gaussian elimination over a general semiring is not guaranteed to terminate. Guaranteed termination can be achieved by restricting the domain (e.g. considering closed semirings instead of arbitrary semirings), or by restricting the type of the matrix (e.g. considering numerical matrices with certain special properties). In the case of numerical matrices, computation of the matrix closure corresponds to matrix inversion: $A^* = (I - A)^{-1}$.

Let A be an $n \times n$ matrix over a semiring. We assume that the closure of a semiring element can be computed in time $O(1)$, whenever this closure exists. Matrix closure A^* can be computed by sequential Gaussian elimination in time $\Theta(n^3)$, provided that the computation terminates. This method is asymptotically optimal for matrices over a general semiring, which can be shown by a standard reduction of the matrix multiplication problem.

The parallel complexity of Gaussian elimination has been extensively studied in many models of parallel computation. A BSP algorithm in [11] works by reducing the problem to the computation of a three-dimensional cube dag (see [11], [16]; many similar algorithms have been proposed earlier in the context of systolic computation). The BSP cost of the cube dag algorithm is $W = O(n^3/p)$, $H = O(n^2/p^{1/2})$, $S = O(p^{1/2})$.

A lower communication cost for computing matrix closure can be achieved by recursive block Gauss–Jordan elimination. This standard method was suggested in [1] as a means of reducing the communication cost of a parallel transitive closure algorithm, which is another special case of matrix closure. The BSP cost of block Gauss–Jordan elimination was analysed in [17]; we summarise the results here for completeness.

For convenience we assume that the resulting matrix A^* must replace the original matrix A. The algorithm works by dividing the matrix into square blocks of size $n/2$,

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \tag{1}$$

and then applying block Gauss–Jordan elimination:

$$\begin{aligned} \bar{A}_{11} &\leftarrow A_{11}^* & \bar{\bar{A}}_{22} &\leftarrow \bar{A}_{22}^* \\ \bar{A}_{12} &\leftarrow \bar{A}_{11} A_{12} & \bar{\bar{A}}_{21} &\leftarrow \bar{\bar{A}}_{22} \bar{A}_{21} \\ \bar{A}_{21} &\leftarrow A_{21} \bar{A}_{11} & \bar{\bar{A}}_{12} &\leftarrow \bar{A}_{12} \bar{\bar{A}}_{22} \\ \bar{A}_{22} &\leftarrow A_{22} \oplus A_{21} \bar{A}_{11} A_{12} & \bar{\bar{A}}_{11} &\leftarrow \bar{A}_{11} \oplus \bar{A}_{21} \bar{\bar{A}}_{22} \bar{A}_{12} \end{aligned} \tag{2}$$

after which every $\bar{\bar{A}}_{ij}$ overwrites A_{ij}. The procedure can be applied recursively to find A_{11}^* and $\bar{A}_{22}^*$. The resulting matrix is

$$A^* = \begin{pmatrix} A_{11}^* \oplus A_{11}^* A_{12} \odot G^* \odot A_{21} A_{11}^* & A_{11}^* A_{12} \odot G^* \\ G^* \odot A_{21} A_{11}^* & G^* \end{pmatrix} \tag{3}$$

where $G = A_{22} \oplus A_{21} A_{11}^* A_{12}$ (here we use both $\odot$ and juxtaposition to denote semiring multiplication). The computation terminates, if all taken closures exist.

The resulting BSP algorithm allows us to trade off the costs of communication and synchronisation in a certain range. In order to account for this tradeoff, we introduce a real parameter α. The algorithm is as follows.

Algorithm 1. *Algebraic path computation.*
Parameters: integer $n \geq p$; real number α, $\alpha_{\min} = 1/2 \leq \alpha \leq 2/3 = \alpha_{\max}$.
Input: $n \times n$ matrix A over a semiring.
Output: $n \times n$ matrix closure A^* (assuming it exists), overwriting A.
Description. The computation is defined by recursion on the size of the matrix. For small blocks, (2) is computed sequentially on an arbitrarily chosen processor. For large blocks, matrix multiplication in (2) is performed by the McColl–Valiant algorithm on all p processors. The details of the algorithm are described in [17].
Cost analysis. The analysis in [17] gives

$$W = O(n^3/p) \qquad H = O(n^2/p^\alpha) \qquad S = O(p^\alpha) \quad \blacksquare$$

For $\alpha = \alpha_{\min} = 1/2$, the cost of Algorithm 1 is $W = O(n^3/p)$, $H = O(n^2/p^{1/2})$, $S = O(p^{1/2})$. This is asymptotically equal to the BSP cost of the cube dag method from [11]. For $\alpha = \alpha_{\max} = 2/3$, the cost of Algorithm 1 is $W = O(n^3/p)$, $H = O(n^2/p^{2/3})$, $S = O(p^{2/3})$. In this case, the communication cost is as low as in matrix multiplication (the McColl–Valiant algorithm). This improvement in communication efficiency is offset by a reduction in synchronisation efficiency. For large n, the communication cost of Algorithm 1 dominates the synchronisation cost, and therefore the communication improvement should

outweigh the loss of synchronisation efficiency. This justifies the use of Algorithm 1 with $\alpha = \alpha_{\max} = 2/3$. Smaller values of α, or the cube dag algorithm, should be considered when the problem is moderately sized.

If the ground semiring is a commutative ring with unit, fast matrix multiplication can be used instead of standard matrix multiplication for computing block products. The BSP cost of the resulting algorithm is

$$W = O(n^\omega/p) \qquad H = O(n^2/p^\alpha) \qquad S = O(p^\alpha)$$

where $1/(\omega - 1) \leq \alpha \leq 2/\omega$.

4 All-Pairs Shortest Paths Computation

4.1 Nonnegative Edge Lengths

In Section 3 we considered the algebraic path problem over an arbitrary semiring. Here we deal with a special case where the semiring is the set of real numbers with ∞, and the numerical operations min and $+$ are used as semiring addition $\oplus$ and multiplication $\odot$ respectively. Since the min operation is idempotent, for all i, j there is a path from i to j of length $A^*[i,j]$ — this is one of the *shortest paths* from i to j. Most algorithms for matrix closure in the $(\min, +)$ semiring can be extended to compute the shortest paths between all pairs of nodes, as well as the distances. Therefore, in this section we use the term *all pairs shortest paths problem* as a synonym for the matrix closure problem in the $(\min, +)$ semiring. Initially, we consider the case where all edge lengths are nonnegative. We then extend our method to general lengths.

The technique of Gauss–Jordan elimination, considered in Section 3, can be applied to the all pairs shortest paths problem. In this context, Gauss–Jordan elimination is commonly known as the *Floyd–Warshall algorithm* (see e.g. [5]). Its block recursive version, identical to Algorithm 1, solves the problem with BSP cost $W = O(n^3/p)$, $H = O(n^2/p^\alpha)$, $S = O(p^\alpha)$, for an arbitrary α, $1/2 \leq \alpha \leq 2/3$.

Alternatively, the problem with nonnegative lengths can be solved by *Dijkstra's algorithm* ([6], see also [5]). This greedy algorithm finds all shortest paths from a fixed source in order of increasing length. The sequential time complexity of Dijkstra's algorithm is $\Theta(n^2)$. To compute the shortest paths between all pairs of nodes in parallel, one can apply Dijkstra's algorithm independently to each node as a source (this approach is suggested e.g. in [10,7]). The resulting algorithm has BSP cost $W = O(n^3/p)$, $H = O(n^2)$, $S = O(1)$. It thus has a higher communication cost, but a lower synchronisation cost, than the Floyd–Warshall algorithm. This tradeoff motivates us to look for an improved BSP algorithm, that would solve the all pairs shortest paths problem efficiently both in communication and synchronisation.

In order to design such an algorithm, we use the principle of *path doubling*. No shortest path may contain more than n edges, therefore $A^n = A^*$. Matrix A^n can be obtained by repeated squaring in $\log n$ matrix multiplications.

Therefore, the local computation cost of computing A^n by repeated squaring is $W = \Theta\big((n^3 \log n)/p\big)$. A refined version of path doubling was proposed in [2,15]. When run in parallel, this method allows one to compute the matrix $A^n = A^*$ with local computation cost $W = O(n^3/p)$. Compared to the Floyd–Warshall algorithm, the new method does not improve on the synchronisation cost by itself; however, an improvement can be achieved by combining the new method with Dijkstra's algorithm.

By a small perturbation of edge lengths, we can always make all edge and path lengths in the graph distinct. Therefore, from now on we assume that all shortest paths are unique. We use the term *path size* for the number of edges in a path. The main idea of the method is to perform path doubling, keeping track not only of path lengths, but also of path sizes. We assume that lengths and sizes are kept in a single data structure, called the *path matrix*. In a such a matrix X, each entry $X[i,j]$ is either ∞, or corresponds to a simple path from i to j. Addition and multiplication of path matrices are defined in the natural way.

For an integer k, let $X(k)$ denote the matrix of all paths in X of size exactly k. More precisely,

$$X(k)[i,j] = \begin{cases} X[i,j] & \text{if path } X[i,j] \text{ has size } k \\ \infty & \text{otherwise} \end{cases}$$

Let $X(k_1, \ldots, k_s) = X(k_1) \oplus \cdots \oplus X(k_s)$ (remembering that $\oplus$ denotes numerical min). Note that for any path matrix X, we have

$$X = X(0, 1, \ldots, m) = X(0) \oplus X(1) \oplus \cdots \oplus X(m)$$

where m is the maximum path size in X.

For path matrices X, Y, we write $X \leq Y$, if $X[i,j] \leq Y[i,j]$ for all i, j (ignoring path sizes). We call an entry $X[i,j]$ *trivial*, if $X[i,j] = \infty$. We call X and Y *disjoint*, if either $X[i,j]$, or $Y[i,j]$ is trivial for all i, j.

Consider the nonnegative all-pairs shortest paths problem defined by path matrix A. This matrix contains all shortest paths of size 0 (the main diagonal) and of size 1 (the off-diagonal entries). For an integer k, matrix A^k contains all shortest paths of size at most k (and maybe some other paths). Suppose that we have computed A^k for some k, $1 \leq k < n$. Our next goal is to compute all shortest paths of size at most $3k/2$. Decompose the path matrix A^k into a disjoint semiring sum:

$$A^k = A^k(0, 1, \ldots, k) = I \oplus A^k(1) \oplus \cdots \oplus A^k(k)$$

Consider the upper half of this sum, which consists of matrices $A^k(k/2 + 1), \ldots, A^k(k)$. The total number of nontrivial entries in all these matrices is at most n^2 (since the matrices are disjoint), hence the average number of nontrivial entries per matrix is at most $2n^2/k$. For some l, $k/2 < l \leq k$, matrix $A^k(l)$ contains at most $2n^2/k$ nontrivial entries. The BSP cost of finding such an l is negligible.

Consider any shortest path of size in the range $l+1,\ldots,3k/2$. This path consists of an initial subpath of size l, and a final subpath of size at most k. Therefore, the semiring sum $A^k \oplus A^k(l) \odot A^k$ contains all shortest paths of size at most $3k/2$:

$$A^k \oplus A^k(l) \odot A^k = \big(I \oplus A^k(l)\big) \odot A^k \leq A^{3k/2}$$

Since $A^k(l)$ has at most $2n^2/k$ nontrivial entries, computation of $A^k(l) \odot A^k$ requires not more than $2n^3/k$ semiring multiplications.

For efficient parallel computation of the sparse-by-dense matrix product $A^k(l) \odot A^k$, we need to partition the problem into p sparse-by-dense matrix multiplication subproblems, where all the sparse arguments have an approximately equal number of nontrivial entries. This can be done by first partitioning the set of rows in $A^k(l)$ into $p^{1/3}/k^{1/3}$ equal subsets, such that each subset contains at most $\frac{2n^2}{k^{2/3} \cdot p^{1/3}}$ nontrivial entries. This partitioning defines, up to a permutation of rows, a decomposition of the matrix into $p^{1/3}/k^{1/3}$ equal horizontal strips. Each strip defines an $\frac{n \cdot k^{1/3}}{p^{1/3}} \times n \times n$ sparse-by-dense matrix multiplication subproblem.

Consider one of the above subproblems. Partition the set of columns in the strip into $p^{1/3}/k^{1/3}$ equal subsets, such that each subset contains at most $\frac{4n^2}{k^{1/3} \cdot p^{2/3}}$ nontrivial entries. This partitioning defines, up to a permutation of columns, a decomposition of the strip into equal square blocks. Each block defines an $\frac{n \cdot k^{1/3}}{p^{1/3}} \times \frac{n \cdot k^{1/3}}{p^{1/3}} \times n$ sparse-by-dense matrix multiplication subproblem. By partitioning the set of columns of the second argument of this subproblem into $p^{1/3} \cdot k^{2/3}$ equal subsets, we obtain $p^{1/3} \cdot k^{2/3}$ sparse-by-dense matrix multiplication subproblems of size $\frac{n \cdot k^{1/3}}{p^{1/3}} \times \frac{n \cdot k^{1/3}}{p^{1/3}} \times \frac{n}{p^{1/3} \cdot k^{2/3}}$.

The total number of resulting sparse-by-dense matrix multiplication subproblems is p. The sparse argument of each subproblem contains at most $\frac{4n^2}{k^{1/3} \cdot p^{2/3}}$ nontrivial entries. The partitioning can be computed by a greedy algorithm, the BSP cost of which is negligible. The BSP cost of computing the matrix product $A^k(l) \odot A^k$ is therefore $W = O\big(n^3/(k \cdot p)\big)$, $H = O\big(n^2/(k^{1/3} \cdot p^{2/3})\big)$, $S = O(1)$.

The path doubling process is stopped after at most $\log_{3/2} p$ rounds, when the matrix A^p (or some matrix $\leq A^p$, which is only better) has been computed. For some q, $1 \leq q \leq p$, matrix $A^p(q)$ contains at most n^2/p nontrivial entries. Therefore, this matrix can be broadcast to every processor with communication cost $H = O(n^2/p)$. Each processor receives the matrix $A^p(q)$, picks n/p nodes, and computes all shortests paths originating in these nodes by n/p independent runs of Dijkstra's algorithm. The result of this computation across all processors is the matrix closure $A^p(q)^*$. Matrix $A^p(q)^*$ contains all shortest paths of sizes that are multiples of q (and maybe some other paths).

Any shortest path in A^* consists of an initial subpath of size that is a multiple of q, and a final subpath of size at most $q \leq p$. Therefore, all shortest paths for the original matrix A can be computed as the matrix product

$$A^p(q)^* \odot A^p = A^*$$

The cost of the resulting algorithm is $W = O(n^3/p)$, $H = O(n^2/p^{2/3})$, $S = O(\log p)$. We can further reduce the synchronisation cost by terminating the path doubling phase after fewer than $\log_{3/2} p$ steps. For $1 \le r \le p^{2/3}$, we can find a q such that the matrix $A^r(q)$ has at most n^2/r nontrivial entries, therefore the communication cost of applying Dijkstra's algorithm to find $A^r(q)^*$ is $H = O(n^2/r)$.

The resulting algorithm is as follows.

Algorithm 2. *All pairs shortest paths (nonnegative case).*
Parameters: integer $n \ge p$; integer r, $1 \le r \le p^{2/3}$.
Input: $n \times n$ matrix A over the $(\min, +)$ semiring of nonnegative real numbers with ∞.
Output: $n \times n$ matrix closure A^*.
Description. The computation proceeds in three stages.

First stage. Compute A^r by at most $\log_{3/2} r$ rounds of path doubling.
Second stage. Select q, $0 < q \le r$, such that $A^r(q)$ contains at most n^2/r nontrivial entries. Broadcast $A^r(q)$ and compute the closure $A^r(q)^*$ by n independent runs of Dijkstra's algorithm, n/p runs per processor.
Third stage. Compute the product $A^r(q)^* \odot A^r = A^*$.

Cost analysis. The local computation and communication costs of the first stage are dominated by the cost of its first round: $W = O(n^3/p)$ and $H = O(n^2/p^{2/3})$. The synchronisation cost of the first stage is $S = O(\log r)$.

The cost of the second stage is $W = O(n^3/p)$, $H = O(n^2/r)$, $S = O(1)$. The cost of the third stage is $W = O(n^3/p)$, $H = O(n^2/p^{2/3})$, $S = O(1)$. The local computation, communication and synchronisation costs of the whole algorithm are

$$W = O(n^3/p) \qquad H = O(n^2/r) \qquad S = O(\log r) \quad \blacksquare$$

The two extremes of Algorithm 2 are the communication-efficient algorithm ($r = p^{2/3}$), with

$$W = O(n^3/p) \qquad H = O(n^2/p^{2/3}) \qquad S = O(\log p)$$

and the multiple Dijkstra algorithm ($r = 1$), with

$$W = O(n^3/p) \qquad H = O(n^2) \qquad S = O(1)$$

The second stage of Algorithm 2 allows the following variation. Instead of using the matrix $A^r(q)$ with at most n^2/r nontrivial entries, we can use the matrix $A^r(r)$. In order to communicate this matrix efficiently, we represent it as a product

$$A^r(r) = A^r(q) \odot A^r(r - q)$$

For some q, $0 \le q < r/2$, the disjoint sum $A^r(q) \oplus A^r(r - q)$ contains at most $2n^2/r$ nontrivial entries. Therefore, the second stage of the algorithm can be

replaced by broadcasting the matrices $A^r(q)$ and $A^r(r-q)$ (or, equivalently, their disjoint sum), recovering the product $A^r(q) \odot A^r(r-q) = A^r(r)$, and computing the closure $A^r(r)^*$. A similar technique of broadcasting a path matrix can be used on every step of path doubling in the first stage of Algorithm 2.

4.2 General Edge Lengths

We now extend the algorithm to graphs where edge lengths may be negative. Formally, the problem consists in finding the closure A^* of a matrix A over the $(\min, +)$ semiring of all real numbers with ∞. The closure exists, if and only if the graph defined by the matrix does not contain a cycle of negative length. We cannot use our original method to solve this more general problem, because Dijkstra's algorithm does not work on graphs with negative edge lengths. However, we can get around this difficulty by replacing Dijkstra's algorithm with an extra stage of sequential path doubling.

The extended algorithm has three stages. In the first stage, we compute the matrix A^{p^2} by $2\log_{3/2} p$ steps of parallel path doubling. Let

$$A^{p^2}((p)) = A^{p^2}(p, 2p, \ldots, p^2)$$

and

$$A^{p^2}((p) - q) = A^{p^2}(p - q, 2p - q, \ldots, p^2 - q)$$

We represent matrix $A^{p^2}((p))$ as a product

$$A^{p^2}((p)) = A^{p^2}(q) \odot A^{p^2}((p) - q)$$

For some q, $0 \leq q < p/2$, the disjoint sum $A^{p^2}(q) \oplus A^{p^2}((p) - q)$ contains at most $2n^2/p$ nontrivial entries. In the second stage, we collect matrices $A^{p^2}(q)$ and $A^{p^2}((p) - q)$ in a single processor, and recover their product $A^{p^2}((p))$. Since the matrix $A^{p^2}((p))$ represents paths of p different sizes $p, 2p, \ldots, p^2$, we can find a size $l \in \{(p/2) \cdot p, (p/2 + 1) \cdot p, \ldots, p^2\}$, such that the matrix $A^{p^2}((p))(l)$ contains at most $2n^2/p$ nontrivial entries. Now the closure $A^{p^2}((p))^* = A^{p^2}(p)^*$ can be computed by sequential path doubling, with the first step computing the semiring sum

$$A^{p^2}((p)) \oplus A^{p^2}((p))(l) \odot A^{p^2}((p)) \leq A^{3p^2/2}((p))$$

The sequential cost of the closure computation is dominated by the cost of its first step, $O(n^3/p)$. In the third stage, it remains to compute the product $A^{p^2}(p)^* \odot A^{p^2} = A^*$.

In contrast with the nonnegative case, early termination of the parallel path doubling phase would increase not only the communication cost, but also the local computation cost. Therefore, we do not consider this option.

The resulting algorithm is as follows.

Algorithm 3. *All pairs shortest paths (general case).*
Parameter: integer $n \geq p$.
Input: $n \times n$ matrix A over the $(\min, +)$ semiring of real numbers with ∞.
Output: $n \times n$ matrix closure A^*.
Description. The computation proceeds in three stages.

First stage. Compute A^{p^2} and $A^{p^2}((p))$ by at most $2\log_{3/2} p$ rounds of path doubling.
Second stage. Select q, $0 < q \leq p/2$, such that the disjoint sum $A^{p^2}(q) \oplus A^{p^2}((p)-q)$ contains at most $2n^2/p$ nontrivial entries. Collect $A^{p^2}(q) \oplus A^{p^2}((p)-q)$ in a single processor, and recover $A^{p^2}((p)) = A^{p^2}(q) \odot A^{p^2}((p)-q)$. Compute the closure $A^{p^2}((p))^* = A^{p^2}(p)^*$ by sequential path doubling.
Third stage. Compute the product $A^{p^2}(p)^* \odot A^{p^2} = A^*$.

Cost analysis. The local computation and communication costs of the first stage are dominated by the cost of its first round: $W = O(n^3/p)$ and $H = O(n^2/p^{2/3})$. The synchronisation cost of the first stage is $S = O(\log p)$.

The local computation cost of the second stage is dominated by the cost of its first round, equal to $W = O(n^3/p)$. The communication and synchronisation costs of the second stage are $H = O(n^2/p)$, $S = O(1)$.

The cost of the third stage is $W = O(n^3/p)$, $H = O(n^2/p^{2/3})$, $S = O(1)$.

The local computation, communication and synchronisation costs of the whole algorithm are

$$W = O(n^3/p) \qquad H = O(n^2/p^{2/3}) \qquad S = O(\log p) \quad \blacksquare$$

The described method is applicable not only to the $(\min, +)$ semiring (the standard shortest paths problem), but also to any semiring where addition is idempotent, e.g. the $(\vee, \wedge)$ semiring (the transitive closure problem), the $(\max, \min)$ semiring (paths of maximum capacity), or the $(\max, \cdot)$ semiring (paths of maximum reliability). Note that in the case of transitive closure computation by Algorithm 2, Boolean matrix multiplication cannot be used instead of general matrix multiplication, since the path doubling process involves the multiplication of path matrices, rather than ordinary Boolean matrices. It is not immediately clear if the BSP cost of general matrix multiplication can be reduced for path matrices with Boolean edge lengths.

5 Conclusions

We have presented a new BSP algorithm for the all-pairs shortest paths problem in a weighted directed dense graph. The algorithm adapts the method of selective path doubling from [2,15] to the BSP framework, and saves a substantial amount of synchronisation by combining selective path doubling with Dijkstra's algorithm. In contrast with the general algebraic path algorithm, which performs $O(p^{1/2})$ to $O(p^{2/3})$ global synchronisation steps, our new algorithm only requires $O(\log p)$ synchronisation steps. The number of synchronisation steps can

Table 1. Summary of presented algorithms

Problem	W	H	S
Matrix multiplication	n^3/p	$n^2/p^{2/3}$	1
Algebraic paths	n^3/p		
general, min H	–	$n^2/p^{2/3}$	$p^{2/3}$
general, min S	–	$n^2/p^{1/2}$	$p^{1/2}$
All-pairs shortest paths	n^3/p		
general	–	$n^2/p^{2/3}$	$\log p$
nonnegative, min H	–	$n^2/p^{2/3}$	$\log p$
nonnegative, min S	–	n^2	1

be further reduced to $O(1)$, if the edge lengths are nonnegative. In this case, the algorithm exhibits a tradeoff between asymptotic costs of communication and synchronisation.

It is not clear yet whether the presented algorithm is practical, because of the significant potential overhead of dealing with path matrices, instead of ordinary numerical matrices (as e.g. in the Floyd–Warshall algorithm). However, our algorithm advances the theoretical understanding of BSP computation on dense graphs, and shows a possible source of faster parallel graph algorithms.

References

[1] A. Aggarwal, A. K. Chandra, and M. Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71(1):3–28, March 1990.

[2] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, April 1997.

[3] B. Carré. *Graphs and Networks*. Oxford Applied Mathematics and Computer Science Series. Clarendon Press, 1979.

[4] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, March 1990.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Electrical Engineering and Computer Science Series. The MIT Press and McGraw–Hill, 1990.

[6] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[7] I. Foster. *Designing and Building Parallel Programs*. Addison–Wesley, 1995.

[8] M. Gondran and M. Minoux. *Graphs and Algorithms*. Wiley—Interscience Series in Discrete Mathematics. John Wiley & Sons, 1984.

[9] M. Gondran and M. Minoux. Linear algebra in dioids: A survey of recent results. *Annals of Discrete Mathematics*, 19:147–164, 1984.

[10] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, January 1977.

[11] W. F. McColl. Scalable computing. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 46–61. Springer-Verlag, 1995.

[12] W. F. McColl. A BSP realisation of Strassen's algorithm. In M. Kara, J. R. Davy, D. Goodeve, and J. Nash, editors, *Abstract Machine Models for Parallel and Distributed Computing*, pages 43–46. IOS Press, 1996.

[13] W. F. McColl. Universal computing. In L. Bougé et al., editors, *Proceedings of Euro-Par '96 (Part I)*, volume 1123 of *Lecture Notes in Computer Science*, pages 25–36. Springer-Verlag, 1996.

[14] G. Rote. Path problems in graphs. *Computing Supplementum*, 7:155–189, 1990.

[15] T. Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica*, 20:309–318, 1998.

[16] A. Tiskin. The bulk-synchronous parallel random access machine. *Theoretical Computer Science*, 196(1–2):109–130, April 1998.

[17] A. Tiskin. Bulk-synchronous parallel Gaussian elimination. In N. N. Vasil'ev and A. M. Vershik, editors, *Representation Theory, Dynamical Systems, Combinatorial and Algorithmic Methods (Part 4)*, volume 258 of *Zapiski Nauchnykh Seminarov POMI*. Russian Academy of Sciences, 1999. Also to appear in *Journal of Mathematical Sciences*.

[18] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

[19] U. Zimmermann. *Linear and Combinatorial Optimization in Ordered Algebraic Structures*, volume 10 of *Annals of Discrete Mathematics*. North-Holland, 1981.

Approximating the Minimum Spanning Tree Weight in Sublinear Time

Bernard Chazelle[1]*, Ronitt Rubinfeld[2], and Luca Trevisan[3]

[1] Princeton University and NEC Research Institute, Princeton, NJ
chazelle@cs.princeton.edu.
[2] NEC Research Institute, Princeton, NJ.
ronitt@research.nj.nec.com.
[3] U.C. Berkeley, Berkeley, CA.
luca@eecs.berkeley.edu.

Abstract. We present a probabilistic algorithm that, given a connected graph G (represented by adjacency lists) of maximum degree d, with edge weights in the set $\{1, \ldots, w\}$, and given a parameter $0 < \varepsilon < 1/2$, estimates in time $O(dw\epsilon^{-2} \log \frac{w}{\epsilon})$ the weight of the minimum spanning tree of G with a relative error of at most ε. Note that the running time does *not* depend on the number of vertices in G. We also prove a nearly matching lower bound of $\Omega(dw\epsilon^{-2})$ on the probe and time complexity of any approximation algorithm for MST weight.

The essential component of our algorithm is a procedure for estimating in time $O(d\varepsilon^{-2} \log \varepsilon^{-1})$ the number of connected components of an unweighted graph to within an additive error of ϵn. The time bound is shown to be tight up to within the $\log \varepsilon^{-1}$ factor. Our connected-components algorithm picks $O(1/\epsilon^2)$ vertices in the graph and then grows "local spanning trees" whose sizes are specified by a stochastic process. From the local information collected in this way, the algorithm is able to infer, with high confidence, an estimate of the number of connected components. We then show how estimates on the number of components in various subgraphs of G can be used to estimate the weight of its MST.

1 Introduction

Traditionally, a linear time algorithm has been held as the gold standard of efficiency. In a wide variety of settings, however, large data sets have become increasingly common, and it is often desirable and sometimes necessary to find very fast algorithms which can assert nontrivial properties of the data in *sublinear* time.

One direction of research that has been suggested is that of property testing [15,8], which relaxes the standard notion of a decision problem. Property testing algorithms distinguish between inputs that have a certain property and

* Part of this research was supported by NSF grant CCR-99817 and ARO Grant DAAH04-96-1-0181.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 190–200, 2001.

those that are far (in terms of Hamming distance, or some other natural distance) from having the property. Sublinear and even constant time algorithms have been designed for testing various algebraic and combinatorial properties (see [14] for a survey). Property testing can be viewed as a natural type of approximation problem and, in fact, many of the property testers have led to very fast, even constant time, approximation schemes for the associated problem (cf. [8,6,7,1]). For example, one can approximate the value of a maximum cut in a dense graph in time $2^{O(\epsilon^{-3}\log 1/\epsilon)}$, with relative error at most ε, by looking at only $O(\epsilon^{-7}\log 1/\epsilon)$ locations in the adjacency matrix [8]. Note that typically such schemes approximate the value of the optimal solution, here the size of a maxcut, without computing the structure that achieves it, i.e., the actual cut. Sometimes, however, a solution can also be constructed in linear or near-linear time.

In this paper, we consider the problem of finding the weight of the minimum spanning tree (MST) of a graph. Finding the MST of a graph has a long and interesting history [3,10,12]. Currently the best known deterministic algorithm of Chazelle [2] runs in $O(m\alpha(m,n))$ time, where n (resp. m) is the number of vertices (resp. edges) and α is inverse-Ackermann, and the randomized algorithm of Karger, Klein and Tarjan [11] runs in linear expected time (see also [5,13] for alternative models).

In this paper, we show that there are conditions under which it is possible to approximate the weight of the MST of a connected graph in time sublinear in the number of edges. We give an algorithm which approximates the MST of a graph G to within a multiplicative factor of $1+\epsilon$ and runs in time $O(dw\epsilon^{-2}\log\frac{w}{\epsilon})$ for any G with max degree d and edge weights in the set $\{1,\ldots,w\}$. The relative error ε $(0<\varepsilon<1/2)$ is specified as an input parameter. Note that if d and ε are constant and the ratios of the edge weights are bounded, then the algorithm runs in constant time. We also extend our algorithm to the case where G has nonintegral weights in the range $[1,w]$, achieving a comparable runtime with a somewhat worse dependence on ϵ.

Our algorithm considers several auxiliary graphs: If G is the weighted graph, let us denote by $G^{(i)}$ the subgraph of G that contains only edges of weight at most i. We estimate the number of connected components in each $G^{(i)}$. To do so, we sample uniformly at random $O(1/\epsilon^2)$ vertices in $G^{(i)}$, and then estimate the size of the component that contains each sampled vertex by constructing "local trees" of some appropriate size defined by a random process. Based on information about these local trees, we can produce a good approximation for the weight of the MST of G. Our algorithm for estimating the number of connected components in a graph runs in time $O(d\varepsilon^{-2}\log\varepsilon^{-1})$ and produces an estimate that is within an additive error of ϵn of the true count. The method is based on a similar principle as the property tester for graph connectivity given by Goldreich and Ron [9].

We give a lower bound of $\Omega(dw/\epsilon^2)$ on the time complexity of any algorithm which approximates the MST weight. In order to prove the lower bound, we give two distributions on weighted graphs, where the support set of one distribution

contains graphs with MST weight at least $1+\epsilon$ times the MST weight of the graphs in the support of the other distribution. We show that any algorithm that reads $o(dw/\epsilon^2)$ weights from the input graph is unlikely to distinguish between graphs from the two distributions. We also prove a lower bound of $O(d/\varepsilon^2)$ on the running time of any approximation algorithm for counting connected components.

2 Estimating the Number of Connected Components

We begin with the problem of estimating the number of components in an arbitrary graph G. We present an algorithm which gives an additive estimate of the number of components in G to within ϵn in $O(d\epsilon^{-2}\log\epsilon^{-1})$ time, for any $0<\varepsilon<1/2$. We later show how to use the ideas from our algorithm to aid in estimating the weight of the MST of a graph.

Let c be the number of connected components in G. Let n_u be the number of vertices in u's component in G. Our algorithm is built around a simple observation:

Fact 1 *Given a graph with vertex set V, for every connected component $I\subseteq V$, $\sum_{u\in I}\frac{1}{n_u}=1$ and $\sum_{u\in V}\frac{1}{n_u}=c$.*

Our strategy is to estimate c by approximating each summand $1/n_u$. Computing n_u directly can take linear time, so we construct an estimator of the quantity $1/n_u$ that has the same expected value. We approximate the number of connected components via the algorithm given in Figure 1. The parameter W is a threshold value, which is set to $2/\varepsilon$ for counting connected components and somewhat higher for its use in MST weight estimation.

```
approx-number-connected-components(G, ε, W)
  uniformly choose r = O(1/ε²) vertices u_1, ..., u_r
  for each vertex u_i,
    set β_i = 0
    take the first step of a BFS from u_i
    (*) flip a coin
        if heads and number of vertices visited in BFS < W
          then resume BFS to double number of visited vertices
            if this allows BFS to complete
              then set β_i = 2^{#coin flips}/#vertices visited in BFS
              else go to (*)
  output ĉ = (n/r) Σ_{i=1}^{r} β_i
```

Fig. 1. Estimating the number of connected components

In the algorithm, doubling the number of vertices does not include duplicate visits to the same vertices; in other words, at each step the number of new vertices visited is supposed to match the number of vertices already visited. In our terminology, the first step of the BFS (shorthand for breadth first search) involves the visit of the single vertex u_i. We now bound the expectation and variance of the estimator β_i for a fixed i. If the BFS from u_i completes, the number of coin flips associated with it is $\lceil \log n_{u_i} \rceil$ and the number of distinct vertices visited is n_{u_i}. Let S denote the set of vertices in components of size $< W$. If $u_i \notin S$, then $\beta_i = 0$; otherwise, it is $2^{\lceil \log n_{u_i} \rceil}/n_{u_i}$ with probability $2^{-\lceil \log n_{u_i} \rceil}$ and 0 otherwise. Since $\beta_i < 2$, the variance of β_i is:

$$\mathbf{var}\,\beta_i \leq \mathbf{E}\,\beta_i^2 \leq 2\mathbf{E}\,\beta_i = \frac{2}{n}\sum_{u\in S}\frac{1}{n_u} \leq \frac{2c}{n}\,.$$

Then the variance of $\hat{c}$ is bounded by

$$\mathbf{var}\,\hat{c} = \mathbf{var}\left(\frac{n}{r}\sum_i \beta_i\right) = \frac{n^2}{r^2}\cdot r \cdot \mathbf{var}\,\beta_i \leq \frac{2nc}{r}. \tag{1}$$

Since the number of components with vertices not in S is at most n/W, we have that

$$c - \frac{n}{W} \leq \mathbf{E}\,\hat{c} = \sum_{u\in S}\frac{1}{n_u} \leq c\,.$$

If we set $W = 2/\epsilon$, then

$$c - \frac{\epsilon n}{2} \leq \mathbf{E}\,\hat{c} \leq c \tag{2}$$

and, by Chebyshev,

$$\mathrm{Prob}[\,|\hat{c} - \mathbf{E}\,\hat{c}| > \epsilon n/2\,] < \frac{\mathbf{var}\,\hat{c}}{(\epsilon n/2)^2} \leq \frac{8c}{\epsilon^2 rn}\,. \tag{3}$$

Choosing $r = O(1/\epsilon^2)$ ensures that with constant probability arbitrarily close to 1, our estimate $\hat{c}$ of the number of connected components deviates from the actual value by at most ϵn.

The expected number of vertices visited in a given execution of the "for loop" is $O(\log W)$, and each newly visited vertex incurs a cost of $O(d)$, so the algorithm runs in expected time $O(d\epsilon^{-2}\log W)$. For our setting of W, this is $O(d\epsilon^{-2}\log\epsilon^{-1})$. As stated, the algorithm's running time is randomized. However, one can get a deterministic running time bound by stopping the algorithm after $Cd\epsilon^{-2}\log\epsilon^{-1}$ steps and outputting 0 if the algorithm has not yet terminated. This event occurs with probability at most $O(1/C)$, which is a negligible addition to the error probability. Thus we have the following theorem:

Theorem 2. *Let c be the number of components in a graph with n vertices. Then Algorithm* approx-number-connected-components *runs in time $O(d\epsilon^{-2}\log\epsilon^{-1})$ and with probability at least $3/4$ outputs $\hat{c}$ such that $|c - \hat{c}| \leq \epsilon n$.*

We can improve the running time to $O((\varepsilon + c/n)d\epsilon^{-2}\log\epsilon^{-1})$, which is much better for small values of c. First, run the algorithm for $r = O(1/\varepsilon)$. By Chebyshev and (1, 2),

$$\text{Prob}\left[\, |\hat{c} - \mathbf{E}\,\hat{c}| > \frac{\mathbf{E}\,\hat{c} + \epsilon n}{2} \,\right] < \frac{8nc}{r(c+\varepsilon n/2)^2} \leq \frac{8n}{r(c+\epsilon n/2)},$$

which is arbitrarily small for $r\varepsilon$ large enough. Next, we use this approximation $\hat{c}$ to "improve" the value of r. We set $r = A/\varepsilon + A\hat{c}/(\varepsilon^2 n)$ for some large enough constant A and we run the algorithm again, with the effect of producing a second estimate c^*. By (2, 3),

$$\text{Prob}[\,|c^* - \mathbf{E}\,c^*| > \epsilon n/2\,] < \frac{8c}{\epsilon^2 rn} \leq \frac{16c}{A\varepsilon n + A\mathbf{E}\,\hat{c}} \leq \frac{16}{A},$$

and so with overwhelming probability, our second estimate c^* of the number of connected components deviates from c by at most ϵn. The running time of this new algorithm is $O((\varepsilon + c/n)d\epsilon^{-2}\log\epsilon^{-1})$.

3 Approximating the Weight of an MST

In this section we present an algorithm for approximating the value of the MST in bounded weight graphs. We are given a connected graph G with maximum degree d and with each edge is assigned an integer weight between 1 and w. We assume that G is represented by adjacency lists or, for that matter, any representation that allows one to access all edges incident to a given vertex in $O(d)$ time. We show how to approximate the weight of the minimum spanning tree of G with a relative error of at most ϵ.

In Section 3.1 we give a new way to characterize the weight of the MST in terms of the number of connected components in subgraphs of G. In Section 3.2 we give the main algorithm and its analysis. Finally, Section 3.3 addresses how to extend the algorithm to the case where G has nonintegral weights.

3.1 MST Weight and Connected Components

We reduce the computation of the MST weight to counting connected components in various subgraphs of G. To motivate the new characterization, consider the special case when G has only edges of weight 1 or 2 (i.e., $w = 2$). Let $G^{(1)}$ be the subgraph of G consisting precisely of the edges of weight 1, and let n_1 be its number of connected components. Then, any MST in G must contain exactly $n_1 - 1$ edges of weight 2, with all the others being of weight 1. Thus, the weight of the MST is exactly $n - 2 + n_1$. We easily generalize this derivation to any w.

For each $0 \leq \ell \leq w$, let $G^{(\ell)}$ denote the subgraph of G consisting of all the edges of weight at most ℓ. Define $c^{(\ell)}$ to be the number of connected components in $G^{(\ell)}$ (with $c^{(0)}$ defined to be n). By our assumption on the weights, $c^{(w)} = 1$. Let $M(G)$ be the weight of the minimum spanning tree of G. Using the above quantities, we give an alternate way of computing the value of $M(G)$:

Claim 3. *For integer $w \geq 2$,*

$$M(G) = n - w + \sum_{i=1}^{w-1} c^{(i)} .$$

Proof: Let α_i be the number of edges of weight i in an MST of G. (Note that α_i is independent of which MST we choose [4].) Observe that for all $0 \leq \ell \leq w-1$, $\sum_{i>\ell} \alpha_i = c^{(\ell)} - 1$, therefore

$$M(G) = \sum_{i=1}^{w} i\alpha_i = \sum_{\ell=0}^{w-1} \sum_{i=\ell+1}^{w} \alpha_i = -w + \sum_{\ell=0}^{w-1} c^{(\ell)} = n - w + \sum_{i=1}^{w-1} c^{(i)}.$$

□

Thus, computing the number of connected components allows us to compute the weight of the MST of G.

3.2 The Main Algorithm

Our algorithm approximates the value of the MST by estimating each of the $c^{(\ell)}$'s. The algorithm is given in Figure 2.

```
approx-MST-weight(G, ε)
  For i = 1, ..., w − 1
    ĉ^(i) = approx-number-connected-components(G^(i), ε, 2w/ε)
  output v̂ = n − w + Σ_{i=1}^{w−1} ĉ^(i)
```

Fig. 2. Approximating the weight of the MST

Theorem 4. *Let v be the weight of the MST of G. Algorithm* approx-mst-weight *runs in time $O(dw\epsilon^{-2} \log \frac{w}{\epsilon})$ and outputs a value $\hat{v}$ that, with probability at least 3/4, differs from v by at most εv.*

Proof: Let $c = \sum_{i=1}^{w-1} c^{(i)}$. Since we call approx-number-connected-components with parameter $W = 2w/\epsilon$, (1, 2) become

$$c^{(i)} - \frac{\epsilon n}{2w} \leq \mathbf{E}\, \hat{c}^{(i)} \leq c^{(i)} \qquad \text{and} \qquad \mathbf{var}\, \hat{c}^{(i)} \leq \frac{2nc^{(i)}}{r} .$$

By summing over i, it follows that $c - \epsilon n/2 \leq \mathbf{E}\, \hat{c} \leq c$ and $\mathbf{var}\, \hat{c} \leq 2nc/r$. Choosing $r\epsilon^2$ large enough, by Chebyshev we have

$$\text{Prob}[\, |\hat{c} - \mathbf{E}\, \hat{c}| > (n - w + c)\epsilon/3 \,] < \frac{18nc}{r\epsilon^2(n - w + c)^2},$$

which is arbitrarily small since we may assume that w/n is sufficiently small (else we might as well compute the MST explicitly, which can be done in $O(dn)$ time [11]). It follows that, with high probability, the error on the estimate satisfies

$$|v - \hat{v}| = |c - \hat{c}| \leq \frac{\varepsilon n}{2} + \frac{\varepsilon(n - w + c)}{3} \leq \varepsilon v.$$

Since the expected running time of each call to approx-number-connected-components is $O(dr \log w/\epsilon)$, the total running time is $O(dw\epsilon^{-2} \log \frac{w}{\epsilon})$. As before, the running time can be made deterministic by stopping execution of the algorithm after $Cdw\epsilon^{-2} \log \frac{w}{\epsilon}$ steps for some appropriately chosen constant C. □

3.3 Nonintegral Weights

Suppose the weights of G are all in the range $[1, w]$, but are not necessarily integral. To extend the algorithm to this case, one can multiply all the weights by $1/\epsilon$ and round each weight to the nearest integer. Then one can run the above algorithm with error parameter $\epsilon/2$ and with a new range of weights $[1, \lceil w/\epsilon \rceil]$ to get a value v. Finally, output ϵv. The relative error introduced by the rounding is at most $\epsilon/2$ per edge in the MST, and hence $\varepsilon/2$ for the whole MST, which gives a total relative error of at most ε. The runtime of the above algorithm is $O(dw\epsilon^{-3} \log \frac{w}{\varepsilon})$.

4 Lower Bounds

We prove that our algorithms for estimating the MST weight and counting connected components are essentially optimal.

Theorem 5. *Any probabilistic algorithm for approximating, with relative error ε, the MST weight of a connected graph with max degree d and weights in $\{1, \ldots, w\}$ requires $\Omega(dw\epsilon^{-2})$ edge weight lookups on average. It is assumed that $w > 1$ and $C\sqrt{w/n} < \varepsilon < 1/2$, for some large enough constant C.*

We can obviously assume that $w > 1$, otherwise the MST weight is always $n - 1$ and no work is required. The lower bound on ε is nonrestrictive since we can always compute the MST exactly in $O(dn)$ time, which is $O(dw\epsilon^{-2})$ for $\varepsilon = O(\sqrt{w/n})$.

Theorem 6. *Given a graph with n vertices, any probabilistic algorithm for approximating the number of connected components with an additive error of εn requires $\Omega(d\epsilon^{-2})$ edge lookups on average. It is assumed that $C/\sqrt{n} < \varepsilon < 1/2$, for some large enough constant C.*

Again, note that the lower bound on ε is nonrestrictive since we can always solve the problem exactly in $O(dn)$ time.

Both proofs revolve around the difficulty of distinguishing between two nearby distributions. For any $0 < q < 1/2$ and $s = 0, 1$, let $\mathcal{D}_q^s$ denote the distribution induced by setting a 0/1 random variable to 1 with probability $q_s = q(1 + (-1)^s\varepsilon)$. We define a distribution $\mathcal{D}$ on n-bit strings as follows: (1) pick $s = 1$ with probability 1/2 (and 0 else); (2) then draw a random string from $\mathcal{D}_q^s$ (by choosing each b_i from $\mathcal{D}_q^s$ independently). Consider a probabilistic algorithm that, given access to such a random bit string, outputs an estimate on the value of s. How well can it do?

Lemma 7. *Any probabilistic algorithm that can guess the value of s with a probability of error below 1/4 requires $\Omega(\varepsilon^{-2}/q)$ bit lookups on average.*

Proof: By Yao's minimax principle, we may assume that the algorithm is deterministic and that the input is distributed according to $\mathcal{D}$. It is intuitively obvious that any algorithm might as well scan $b_1 b_2 \cdots$ until it decides it has seen enough to produce an estimate of s. In other words, there is no need to be adaptive in the choice of bit indices to probe (but the running time itself can be adaptive). To see why is easy. An algorithm can be modeled as a binary tree with a bit index at each node and a 0/1 label at each edge. An adaptive algorithm may have an arbitrary set of bit indices at the nodes, although we can assume that the same index does not appear twice along any path. Each leaf is naturally associated with a probability, which is that of a random input from $\mathcal{D}$ following the path to that leaf. The performance of the algorithm is entirely determined by these probabilities and the corresponding estimates of s. Because of the independence of the random b_i's, we can relabel the tree so that each path is a prefix of the same sequence of bit probes $b_1 b_2 \cdots$. This oblivious algorithm has the same performance as the adaptive one.

We can go one step further and assume that the running time is the same for all inputs. Let t^* be the expected number of probes, and let $0 < \alpha < 1$ be a small constant. With probability at most α, a random input takes time $\geq t \stackrel{\text{def}}{=} t^*/\alpha$. Suppose that the prefix of bits examined by the algorithm is $b_1 \cdots b_u$. If $u < t$, simply go on probing $b_{u+1} \cdots b_t$ without changing the outcome. If $u > t$, then stop at b_t and output $s = 1$. Thus, by adding α to the probability of error, we can assume that the algorithm consists of looking up $b_1 \cdots b_t$ regardless of the input string.

Let $p_s(b_1 \cdots b_t)$ be the probability that a random t-bit string chosen from $\mathcal{D}_q^s$ is equal to $b_1 \cdots b_t$. The probability of error satisfies

$$p_{\text{err}} \geq \frac{1}{2} \sum_{b_1 \cdots b_t} \min_s p_s(b_1 \cdots b_t).$$

Obviously, $p_s(b_1 \cdots b_t)$ depends only on the number of ones in the string, so if $p_s(k)$ denotes the probability that $b_1 + \cdots + b_t = k$, then

$$p_{\text{err}} \geq \frac{1}{2} \sum_{k=0}^{t} \min_s p_s(k). \tag{4}$$

By the normal approximation of the binomial distribution,

$$p_s(k) \to \frac{1}{\sqrt{2\pi t q_s(1-q_s)}}\, e^{-\frac{(k-tq_s)^2}{2tq_s(1-q_s)}},$$

as $t \to \infty$. This shows that $p_s(k) = \Omega(1/\sqrt{qt}\,)$ over an interval I_s of length $\Omega(\sqrt{qt}\,)$ centered at tq_s. If $qt\varepsilon^2$ is smaller than a suitable constant γ_0, then $|tq_0 - tq_1|$ is small enough that $I_0 \cap I_1$ is itself an interval of length $\Omega(\sqrt{qt}\,)$; therefore $p_{\text{err}} = \Omega(1)$. This shows that if the algorithm runs in expected time $\gamma_0\varepsilon^{-2}/q$, for some constant $\gamma_0 > 0$ small enough, then it will fail with probability at least some absolute constant. By setting α small enough, we can make that constant larger than 2α. This means that, prior to uniformizing the running time, the algorithm must still fail with probability α.

Note that by choosing γ_0 small enough, we can always assume that $\alpha > 1/4$. Indeed, suppose by contradiction that even for an extremely small γ_1, there is an algorithm that runs in time at most $\gamma_1\varepsilon^{-2}/q$ and fails with probability $\leq 1/4$. Then run the algorithm many times and take a majority vote. In this way we can bring the failure probability below α for a suitable $\gamma_1 = \gamma_1(\alpha, \gamma_0) < \gamma_0$, and therefore reach a contradiction. This means that an expected time lower than ε^{-2}/q by a large enough constant factor causes a probability of error at least $1/4$. □

Proof (Theorem 6): Consider the graph G consisting of a simple cycle of n vertices $v_1, \ldots, v_n$. Pick $s \in \{0,1\}$ at random and take a random n-bit string $b_1 \cdots b_n$ with bits drawn independently from $\mathcal{D}^s_{1/2}$. Next, remove from G any edge $(v_i, v_{i+1 \bmod n})$ if $b_i = 0$. Because $\varepsilon > C/\sqrt{n}$, the standard deviation of the number of components, which is $\Theta(\sqrt{n}\,)$, is sufficiently smaller than εn so that with overwhelming probability any two graphs derived from $\mathcal{D}^0_{1/2}$ and $\mathcal{D}^1_{1/2}$ differ by more than $\varepsilon n/2$ in their numbers of connected components. That means that any probabilistic algorithm that estimates the number of connected components with an additive error of $\varepsilon n/2$ can be used to identify the correct s. By Lemma 7, this requires $\Omega(\varepsilon^{-2})$ edge probes into G on average. Replacing ε by 2ε proves Theorem 6 for graphs of degree $d = 2$. For arbitrary d, we may simply add $d-2$ loops to each vertex. Each linked list thus consists of two "cycle" pointers and $d-2$ "loop" ones. If we place the cycle pointers at random among the loop ones, then it takes $\Omega(d)$ probes on average to hit a cycle pointer. If we single out the probes involving cycle pointers, it is not hard to argue that the probes involving cycle pointers are, alone, sufficient to solve the connected components problem on the graph deprived of its loops: one expects at most $O(T/d)$ such probes and therefore $T = \Omega(d\varepsilon^{-2})$. □

Proof (Theorem 5): Again we begin with the case $d = 2$. The input graph G is a simple path of n vertices. Pick $s \in \{0,1\}$ at random and take a random $(n-1)$-bit string $b_1 \cdots b_{n-1}$ with bits drawn independently from $\mathcal{D}^s_q$, where $q = 1/w$. Assign weight w (resp. 1) to the i-th edge along the path if $b_i = 1$ (resp. 0). The MST of G has weight $n - 1 + (w-1)\sum b_i$, and so its expectation is $\Theta(n)$. Also,

note that the difference Δ in expectations between drawing from $\mathcal{D}_q^0$ or $\mathcal{D}_q^1$ is $\Theta(\varepsilon n)$.

Because $\varepsilon > C\sqrt{w/n}$, the standard deviation of the MST weight, which is $\Theta(\sqrt{nw}\,)$, is sufficiently smaller than Δ that with overwhelming probability any two graphs derived from $\mathcal{D}_q^0$ and $\mathcal{D}_q^1$ differ by more than $\Delta/2$ in MST weight. Therefore, any probabilistic algorithm that estimates the weight with a relative error of ε/D, for some large enough constant D, can be used to identify the correct s. By Lemma 7, this means that $\Omega(w\varepsilon^{-2})$ probes into G are required on average.

For $d > 2$, simply join each vertex in the cycle to $d-2$ others (say, at distance > 2 to avoid introducing multiple edges) and, as usual, randomize the ordering in each linked list. Assign weight $w+1$ to the new edges. (Allowing the maximum weight to be $w+1$ instead of w has no influence on the lower bound we are aiming for.) Clearly none of the new edges are used in the MST, so the problem is the same as before, except that we now have to find our way amidst $d-2$ spurious edges, which takes the complexity to $\Omega(dw\varepsilon^{-2})$. □

5 Open Questions

It is natural to ask what can be done if the max degree restriction is lifted. We have made some progress on the case of graphs of bounded *mean* degree. Our algorithm for the case of nonintegral weights requires extra time. Is this necessary? Can the ideas in this paper be extended to finding maximum weighted independent sets in general matroids? There are now a small number of examples of approximation problems that can be solved in sublinear time; what other problems lend themselves to sublinear approximation schemes? More generally, it would be interesting to gain a more global understanding of what can and cannot be approximated in sublinear time.

References

[1] Alon, N., Dar, S., Parnas, M., Ron, D., *Testing of clustering,* Proc. *FOCS*, 2000.
[2] Chazelle, B., *A minimum spanning tree algorithm with inverse-Ackermann type complexity*, J. ACM, 47 (2000), 1028–1047.
[3] Chazelle, B., *The Discrepancy Method: Randomness and Complexity*, Cambridge University Press, 2000.
[4] Eppstein, D., *Representing all minimum spanning trees with applications to counting and generation,* Tech. Rep. 95-50, ICS, UCI, 1995.
[5] Fredman, M.L., Willard, D.E. *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. Comput. and System Sci., 48 (1993), 424–436.
[6] Frieze, A., Kannan, R. *Quick approximation to matrices and applications,* Combinatorica, 19 (1999).
[7] Frieze, A., Kannan, R., Vempala, S., *Fast monte-carlo algorithms for finding low-rank approximations*, Proc. 39th FOCS (1998).

[8] Goldreich, O., Goldwasser, S., Ron, D., *Property testing and its connection to learning and approximation*, Proc. 37th FOCS (1996), 339–348.
[9] Goldreich, O., Ron, D., *Property testing in bounded degree graphs*, Proc. 29th STOC (1997), 406–415.
[10] Graham, R.L., Hell, P. On the history of the minimum spanning tree problem, *Ann. Hist. Comput.* 7 (1985), 43–57.
[11] Karger, D.R., Klein, P.N, Tarjan, R.E., *A randomized linear-time algorithm to find minimum spanning trees,* J. ACM, 42 (1995), 321–328.
[12] Nešetřil, J. A few remarks on the history of MST-problem, *Archivum Mathematicum, Brno* 33 (1997), 15–22. Prelim. version in *KAM Series,* Charles University, Prague, No. 97–338, 1997.
[13] Pettie, S., Ramachandran, V. *An optimal minimum spanning tree algorithm*, Proc. 27th ICALP (2000).
[14] Ron, D., *Property testing (a tutorial),* to appear in "Handbook on Randomization."
[15] Rubinfeld, R., Sudan, M., *Robust characterizations of polynomials with applications to program testing,* SIAM J. Comput. 25 (1996), 252–271.

Approximation Hardness of TSP with Bounded Metrics

Lars Engebretsen[1,⋆] and Marek Karpinski[2,⋆⋆]

[1] MIT Laboratory for Computer Science
200 Technology Square, NE43-369
Cambridge, Massachusetts 02139-3594
enge@mit.edu
[2] Department of Computer Science
University of Bonn
53117 Bonn
marek@cs.uni-bonn.de

Abstract. The general asymmetric (and metric) TSP is known to be approximable only to within an $O(\log n)$ factor, and is also known to be approximable within a constant factor as soon as the metric is bounded. In this paper we study the asymmetric and symmetric TSP problems with bounded metrics and prove approximation lower bounds of 101/100 and 203/202, respectively, for these problems. We prove also approximation lower bounds of 321/320 and 743/742 for the asymmetric and symmetric TSP with distances one and two.

1 Introduction

A common special case of the Traveling Salesman Problem (TSP) is the metric TSP, where the distances between the cities satisfy the triangle inequality. The decision version of this special case was shown to be **NP**-complete by Karp [9], which means that we have little hope of computing exact solutions in polynomial time. Christofides [5] has constructed an elegant algorithm approximating the metric TSP within 3/2, i.e., an algorithm that always produces a tour whose weight is at most a factor 3/2 from the weight of the optimal tour. For the case when the distance function may be asymmetric, the best known algorithm approximates the solution within $O(\log n)$, where n is the number of cities [7]. As for lower bounds, Papadimitriou and Yannakakis [12] have shown that there exists some constant, see also [1], such that it is **NP**-hard to approximate the TSP where the distances are constrained to be either one or two—note that such a distance function always satisfies the triangle inequality—within that constant. This lower bound was improved by Engebretsen [6] to $2805/2804 - \epsilon$ for the asymmetric and $5381/5380 - \epsilon$ for the symmetric, respectively, TSP with distances one and two. Böckenhauer et. al [3,4] considered the symmetric TSP

⋆ Supported by the Marcus Wallenberg Foundation.
⋆⋆ Supported in part by DFG grant, DIMACS, and IST grant 14036 (RAND-APX).

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 201–212, 2001.

with distances one, two and three, and were able to prove a lower bound of $3813/3812 - \epsilon$. (For a discussion of bounded metric TSP, see also Trevisan [13].) It appears that the metric TSP lacks the good definability properties which were needed (so far) for proving strong nonapproximability results. Therefore, any new insights into explicit lower bounds here seem to be of a considerable interest.

Papadimitriou and Vempala [10] recently announced lower bounds of $42/41 - \epsilon$ and $129/128 - \epsilon$, respectively, for the asymmetric and symmetric versions, respectively, of the TSP with graph metric, but left the question of the approximability for the case with bounded metric open. However, their proof contained an error influencing the explicit constants. The corrected proof and the new constants of $98/97 - \epsilon$ and $234/233 - \epsilon$ are computed in [11]. Apart from being an interesting question on its own, it is conceivable that the special cases with bounded metric are easier to approximate than the cases when the distance between two points can grow with the number of cities in the instance. Indeed, the asymmetric TSP with distances bounded by B can be approximated within B by just picking any tour as the solution and the asymmetric TSP with distances one and two can be approximated within $17/12$ [14]. The symmetric version of the latter problem can be approximated within $7/6$ [12].

In this paper, we consider the case when the metric contains only integer distances between one and eight and prove a lower bound of $101/100 - \epsilon$ for the asymmetric case and $203/202 - \epsilon$ for the symmetric case. This is an improvement of an order of magnitude compared to the previous best known bounds of $2805/2804 - \epsilon$ and $3813/3812 - \epsilon$ for this case, respectively [3,4,6]. We also prove that it is **NP**-hard to approximate the asymmetric TSP with distances one and two within $321/320 - \epsilon$, for any constant $\epsilon > 0$. For the symmetric version of the latter problem we show a lower bound of $743/742 - \epsilon$. The previously best known bounds for this case are $2805/2804 - \epsilon$ and $5381/5380 - \epsilon$, respectively [6]. Our proofs depend on explicit reductions from certain bounded dependency instances of linear equations satisfiability. The main idea is to construct certain uniform circles of equation gadgets and, in the second part, certain combined hybrid circle constructions. The reductions for the symmetric case are omitted from this extended abstract; they will appear in the full version of the paper.

Definition 1. *The* Asymmetric Traveling Salesman Problem (ATSP) *is the following minimization problem: Given a collection of cities and a matrix whose entries are interpreted as the distance from a city to another, find the shortest tour starting and ending in the same city and visiting every city exactly once.*

Definition 2. (1,B)-ATSP *is the special case of ATSP where the entries in the distance matrix obey the triangle inequality and the off-diagonal entries in the distance matrix are integers between 1 and B.*

2 The Hardness of (1,B)-ATSP

We reduce, similarly to Papadimitriou and Vempala [10,11], from Håstad's lower bound for E3-Lin mod 2 [8]. Our construction consists of a circle of *equation*

gadgets testing odd parity. This is no restriction since we can easily transform a test for even parity into a test for odd parity by flipping a literal. Three of the edges in the equation gadget correspond to the variables involved in the parity check. These edges are in fact gadgets, so called *edge gadgets*, themselves. Edge gadgets from different equation gadgets are connected to ensure consistency among the edges representing a literal. This requires the number of negative occurrences of a variable to be equal to the number of positive occurrences. This is no restriction since we can duplicate every equation a constant number of times and flip literals to reach this property.

Definition 3. E3-Lin mod 2 *is the following maximization problem: Given an instance of n variables and m equations over $\boldsymbol{Z}_2$ with exactly three unknowns in each equation, find an assignment to the variables that satisfies as many equations as possible.*

Theorem 1 ([8]). *There exists instances of E3-Lin mod 2 with $2m$ equations such that, for any constant $\epsilon > 0$, it is **NP**-hard to decide if at most ϵm or at least $(1-\epsilon)m$ equations are left unsatisfied by the optimal assignment. Each variable in the instance occurs a constant number of times.*

We describe our instance of (1,B)-ATSP by constructing a weighted directed graph G and then let the (1,B)-ATSP instance have the nodes of G as cities. For two nodes u and v in G, let $\ell(u,v)$ be the length of the shortest path from u to v in G. The distance between two cities u and v is the (1,B)-ATSP instance is then defined to be $\min\{B, \ell(u,v)\}$.

2.1 The Gadgets

The gadgets are parametrized by the parameters a, b and d; they will be specified later. Our construction follows Papadimitriou and Vempala [10,11], but we use a slightly different accounting method in our proofs.

The equation gadget for equations of the form $x+y+z=0$ is shown in Fig. 1. The key property of this gadget is that there is a Hamiltonian path through the gadget only if zero or two of the ticked edges are traversed. To form the circle of equation gadgets, vertex A in one gadget coincides with vertex B in another gadget.

The edge gadget is shown in Fig. 2. Each of the bridges is shared between two different edge gadgets, one corresponding to a positive occurrence of the literal and one corresponding to a negative occurrence. The precise coupling is provided by a perfect matching in a d-regular bipartite multigraph $(V_1 \cup V_2, E)$ on $2k$ vertices with the following property: For any partition of V_1 into subsets S_1, U_1 and T_1 and any partition of V_2 into subsets S_2, U_2 and T_2 such that there are no edges from U_1 to U_2, the total number of edges from vertices in T_1 to vertices in T_2 is greater than

$$\frac{\min\bigl\{k, |U_1|+|T_2|+|S_1|+|S_2|, |U_2|+|T_1|+|S_1|+|S_2|\bigr\}}{a+b} - \frac{|S_1|+|S_2|}{2}\,.$$

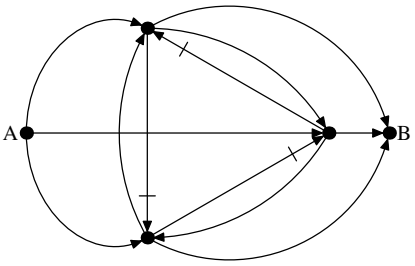


Fig. 1. The gadget for equations of the form $x + y + z = 0$. There is a Hamiltonian path from A to B only if zero or two of the ticked edges, which are actually gadgets themselves (Fig. 2), are traversed. The non-ticked edges have weight 1.

The purpose of this construction is to ensure that it is always optimal for the tour to traverse the graph in such a way that all variables are given consistent values. The edge gadget gives an assignment to an occurrence of a variable by the way it is traversed.

Definition 4. *We call an edge gadget where all bridges are traversed from left to right in Fig. 2* traversed *and an edge gadget where all bridges are traversed from right to left* untraversed. *All other edge gadgets are called* semitraversed.

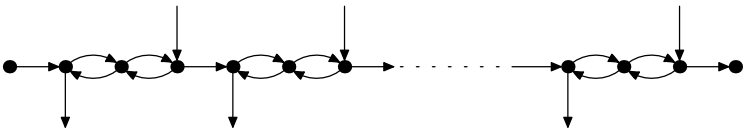


Fig. 2. The edge gadget consists of d bridges. Each of the bridges are shared between two different edge gadgets and consist of two undirected edges of weight $a/2$. The leftmost directed edge above has weight 1, the directed edges leaving a bridge have weight b.

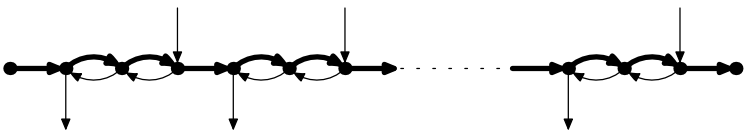


Fig. 3. A traversed edge gadget represents the value 1.

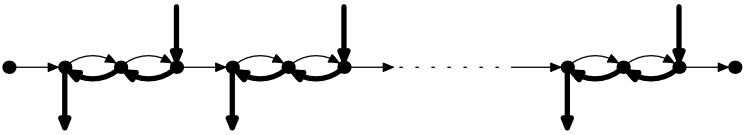


Fig. 4. An untraversed edge gadget represents the value 0.

2.2 Proof of Correctness

If we assume that the tour behaves nicely, i.e., that the edge gadgets are either traversed or untraversed, it is straightforward to establish a correspondence between the length of the tour and the number of unsatisfied equations.

Lemma 1. *The only way to traverse the equation gadget in Fig. 1 with a tour of length* 4*—if the edge gadgets count as length one for the moment—is to traverse an even number of edge gadgets. All other locally optimal traversals have length* 5*.*

Proof. It is easy to see that any tour traversing two ticked edges and leaving the third one untraversed has length 4. Any tour traversing one ticked edge and leaving the other two ticked edges untraversed has length at least 5. Strictly speaking, it is impossible to have three traversals since this does not result in a tour. However, we can regard the case when the tour leaves the edge gadget by jumping directly to the exit node of the equation gadget as a tour with three traversals; such a tour gives a cost of 5.

Lemma 2. *In addition to the length* 1 *attributed to the edge gadget above, the length of a tour traversing an edge gadget in the intended way is* $d(a + b)$*.*

Proof. Each bridge has length a, and every bridge must have one of the outgoing edge traversed. Thus, the total cost is $d(a + b)$.

Lemma 3. *Suppose that there are* $2m$ *equations in the E3-Lin mod 2 instance. If the tour is shaped in the intended way, i.e., every edge gadget is either traversed or untraversed, the length of the tour is* $3md(a + b) + 4m + u$*, where* u *is the number of unsatisfied equations resulting from the assignment represented by the tour.*

Proof. The length of the tour on an edge gadgets is $d(a+b)$. There are three edge gadgets corresponding to every equation and every bridge in the edge gadget is shared between two equation gadgets. Thus, the length of the tour on the edge gadgets is $2m \cdot 3d(a + b)/2 = 3md(a + b)$ The length of the tour on an equation gadget is 4 if the equation is satisfied and 5 otherwise. Thus, the total length is $3md(a + b) + 4m + u$.

The main challenge now is to prove that the above correspondence between the length of the optimum tour and the number of unsatisfied equation holds also when we drop the assumption that the tour is shaped in the intended way.

To count the excessive cost due to traversed non-edges of the graph defining our (1,B)-ATSP instance, we note that every traversed non-edge of weight $w > 1$ corresponds to a path of length $\min\{w, B\}$ on edges in the graph defining the instance. We thus reroute every such tour its corresponding path if $w \leq B$; if $w > B$ we make the tour follow the first $B/2$ and last $B/2$ edges of the tour and then pretend that the tour does a jump of zero cost between these two vertices. This produces something which is not a tour—we call it a *pseudo-tour*—since some edges are traversed more than once and some vertices are connected to more than two traversed edges. From now on, most of the reasoning concerns this pseudo-tour. Our proof uses the following technical lemma:

Lemma 4 ([11]). *For k sufficiently large, almost every 4-regular bipartite multigraph $(V_1 \cup V_2, E)$ on $2k$ vertices has the following property: For any partition of V_1 into subsets S_1, U_1 and T_1 and any partition of V_2 into subsets S_2, U_2 and T_2 such that there are no edges from U_1 to U_2, the total number of edges from vertices in T_1 to vertices in T_2 is greater than*

$$\frac{\min\bigl\{k, |U_1| + |T_2| + |S_1| + |S_2|, |U_2| + |T_1| + |S_1| + |S_2|\bigr\}}{8} - \frac{|S_1| + |S_2|}{2}.$$

Given the above lemma, the following sequence of lemmas give a lower bound on the extra cost, not counting the "normal" cost of $d(a + b)$ per edge gadget and 4 per equation gadget, that results from a non-standard behavior of the tour.

We have already seen that an unsatisfied equation adds an extra cost of 1. Edge gadgets that are either traversed or untraversed do not add any extra cost, except for the case when two traversed equation gadgets share a bridge; this results in a bridge being traversed in both directions by the pseudo-tour. A pseudo-tour resulting from a proper TSP tour can never result in two untraversed edge gadgets sharing a bridge; this would imply a cycle of length $2a$ in the original TSP tour.

Lemma 5. *Two traversed edge gadgets that share a bridge give an extra cost of $a + b$ to the length of the tour.*

Proof. If two traversed edge gadgets are connected, there must be a bridge that is traversed in both directions. Such a bridge gives an extra cost of $a + b$.

Lemma 6. *Suppose that $B \geq 2\max\{a, b\}$. Then every semitraversed edge gadget adds an extra cost of at least $\min\{a, b\}$ to the length of the tour.*

Proof (sketch). We call a bridge balanced with respect to a pseudo-tour if there is at least one edge of the pseudo-tour adjacent to each endpoint of the bridge. Note that an unbalanced bridge always gives an extra cost of a, since the bridge must be traversed in both directions by the pseudo-tour. Thus, we always obtain an extra cost of two if any of the bridges are unbalanced.

Now assume that all bridges are balanced. Since the edge gadget is semitraversed, all bridges cannot be traversed in the same direction. Thus, there are two adjacent bridges that are traversed in different directions. This gives an extra cost of b.

Lemma 7. *For $a = b = d = 4$, there exists a coupling of the equation gadgets with the property that it can never be advantageous to have inconsistently traversed equation gadgets.*

Proof. Repeat the following argument for every variable x:

Let k be the number of occurrences of x (and also the number of occurrences of $\bar{x}$). Pick a bipartite graph on $2k$ vertices having the properties stated in Lemma 4. We know by Lemma 4 that such a graph exists—since the graph has constant size, we can try all possible graphs in constant time.

Put occurrences of x at one side and occurrences of $\bar{x}$ on the other side of the bipartite graph. Each vertex in the graph can be labeled as T, U or S, depending on whether it is traversed, untraversed or semitraversed. Let T_1 be the set of traversed positive occurrences and T_2 be the set of traversed negative occurrences. Define U_1, U_2, S_1, and S_2 similarly. We can assume that $|U_1|+|T_2| \leq |U_2|+|T_1|$—otherwise we just change the indexing convention.

We now consider a modified tour where the positive occurrences are traversed and the negative occurrences are untraversed. This decreases the cost of tour by at least $4(|S_1|+|S_2|)+8|(T_1,T_2)|$, where $|(T_1,T_2)|$ denotes the number of edges between T_1 and T_2, and increases it by $\min\{k, |S_1|+|S_2|+|U_1|+|T_2|\}$. But the bipartite graph has the property that

$$8\big|(T_1,T_2)\big| \geq \min\big\{k, |U_1|+|T_2|+|S_1|+|S_2|\big\} - 4\big(|S_1|+|S_2|\big)$$

which implies that the cost of tour decreases by this transformation. Thus, we can assume that x is given a consistent assignment by the tour.

Theorem 2. *For any constant $\epsilon > 0$, it is **NP**-hard to approximate (1,8)-ATSP within $101/100 - \epsilon$.*

Proof. Given an instance of E3-Lin mod 2 with $2m$ equations where every variable occurs a constant number of times, we construct the corresponding instance of (1,8)-ATSP with $a = b = d = 4$. This can be done in polynomial time. By the above lemma, we can assume that all edge gadgets are traversed consistently in this instance. The assignment obtained from this traversal satisfies $2m - u$ equations if the length of the tour is $3md(a+b)+4m+u$. If we could decide if the length of the optimum tour is at most $(3d(a+b)+4+\epsilon_1)m$ or at least $(3d(a+b)+5-\epsilon_2)m$, we could decide if at most $\epsilon_1 m$ or at least $(1-\epsilon_2)m$ of the equations are left unsatisfied by the corresponding assignment. But to decide this is **NP**-hard by Theorem 1.

In the full version of this paper, we also prove the following theorem:

Theorem 3. *For any constant $\epsilon > 0$, it is **NP**-hard to approximate (1,8)-TSP within $203/202 - \epsilon$.*

3 The Hardness of (1,2)-ATSP

We apply the construction used by Berman and Karpinski [2] to prove stronger hardness results for instances of several combinatorial optimization problems where the number of occurrences of every variable is bounded by some constant. In particular, [2] devises a reduction from systems of linear equations mod two with exactly three unknowns in each equation to a problem called *Hybrid* with the two following properties: Each equation contains either two or three literals and each literal occurs exactly three times.

Definition 5. Hybrid *is the following maximization problem: Given a system of linear equations mod* 2 *containing* n *variables,* m_2 *equations with exactly two unknowns, and* m_3 *equations exactly with three unknowns, find an assignment to the variables that satisfies as many equations as possible.*

Theorem 4 ([2]). *There exists instances of Hybrid with* 42ν *variables,* 60ν *equations with two variables, and* 2ν *equations with three variables such that:*

1. *Each variable occurs exactly three times.*
2. *For any constant* $\epsilon > 0$*, it is* ***NP****-hard to decide if at most* $\epsilon\nu$ *or at least* $(1-\epsilon)\nu$ *equations are left unsatisfied.*

Since we adopt the construction of Berman and Karpinski [2], we can partly rely on their main technical lemmas, which simplifies our proof of correctness.

On a high level, the (1,2)-ATSP instance in our reduction consists of a circle formed by *equation gadgets* representing equations of the form $x + y + z = 0$ and $x + y = 1$. These gadgets are coupled in a way ensuring that the three occurrences of a variable are given consistent values. In fact, the instances of Hybrid produced by the Berman-Karpinski construction have a very special structure. Every variable occurs in at least two equations with two unknowns, and those equations are all equivalences, i.e., equations of the form $x + y = 0$. Since our gadget for equations with two unknowns tests odd parity, we have to rewrite those equations as $x + \bar{y} = 1$ instead. Similarly, the equations of the form $x + y + z = 1$ must be rewritten with one variable negated since our gadgets for equations with three unknowns only test even parity. Turning to the coupling needed to ensure consistency, we have three occurrences of every variable. Since we do not have any gadgets testing odd parity for three variables or even parity for two variables, we may have to negate some of the occurrences. We now argue that there are either one or two negated occurrences of every variable. The Hybrid instance produced by the Berman-Karpinski construction can be viewed as a collection of wheels where the nodes correspond to variables and edges to equations. The edges within a wheel all represent equations with two unknowns, while the equations with three unknowns are represented by hyperedges connecting three different wheels. The equations corresponding to the edges forming the perimeter of the wheel can be written as $x_1 + \bar{x}_2 = 1$, $x_2 + \bar{x}_3 = 1$, $\ldots$, $x_{k-1} + \bar{x}_k = 1$, and $x_k + \bar{x}_1 = 1$, which implies that there is at least one negated and at least one unnegated occurrence of each variable.

Corollary 1. *There exists instances of Hybrid with* 42ν *variables,* 60ν *equations of the form* $x + \bar{y} = 1 \bmod 2$*, and* 2ν *equations of the form* $x + y + z = 0 \bmod 2$ *or* $x + y + \bar{z} = 0 \bmod 2$ *such that:*

1. *Each variable occurs exactly three times.*
2. *There is at least one positive and at least one negative occurrence of each variable.*
3. *For any constant* $\epsilon > 0$*, it is* ***NP****-hard to decide if at most* $\epsilon\nu$ *or at least* $(1 - \epsilon)\nu$ *equations are left unsatisfied.*

To prove our hardness result for (1,2)-ATSP, we reduce instances of Hybrid of the form described in Corollary 1 to instances of (1,2)-ATSP and prove that, given a tour in the (1,2)-ATSP instance, it is possible to construct an assignment to the variables in the original Hybrid instance with the property that the number of unsatisfied equations in the Hybrid instance is related to the length of the tour in the (1,2)-ATSP instance.

To describe a (1,2)-TSP instance, it is enough to specify the edges of weight one. We do this by constructing a graph G and then let the (1,2)-TSP instance have the nodes of G as cities. The distance between two cities u and v is defined to be one if (u, v) is an edge in G and two otherwise. To compute the weight of a tour, it is enough to study the parts of the tour traversing edges of G. In the asymmetric case G is a directed graph.

Definition 6. *We call a node where the tour leaves or enters* G *an* endpoint*. A node with the property that the tour both enters and leaves* G *in that particular node is called a* double endpoint *and counts as two endpoints.*

If c is the number of cities and $2e$ is the total number of endpoints, the weight of the tour is $c + e$ since every edge of weight two corresponds to two endpoints.

3.1 The Gadgets

The equation gadget for equations of the form $x + y + z = 0$ is shown in Fig. 1—the same gadget as in the $(1,B)$ case. However, the ticked edges now represent a different structure.

The equation gadget for equations of the form $x + y = 1$ is shown in Fig. 5. The key property of this gadget is that there is a Hamiltonian path through the gadget only if one of the ticked edges is traversed.

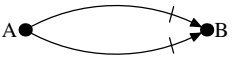


Fig. 5. The gadget for equations of the form $x + y = 1$. There is a Hamiltonian path from A to B only if one of the ticked edges is traversed.

The ticked edges in the equation gadgets are syntactic sugar for a construction ensuring consistency among the three occurrences of each variable. As we

noted above, either one or two of the occurrences of a variable are negated. The construction in Fig. 6 ensures that the occurrences are given consistent values, i.e., that either $x = 0$ and $\bar{x} = 1$, or $x = 1$ and $\bar{x} = 0$. If there is one negated occurrence of a variable, the upper part of the gadget connects with that occurrence and the lower part connects with the two unnegated occurrences. If there are two negated occurrences, the situation is reversed.

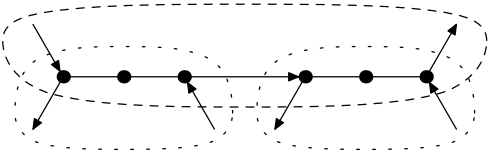


Fig. 6. The gadget ensuring consistency for a variable. If there are two positive occurrences of the variable, the ticked edges corresponding to those occurrences are represented by the parts enclosed in the dotted curves and the ticked edge corresponding to the negative occurrence is represented by the part enclosed in the dashed curve. If there are two negative occurrences, the rôles are reversed.

3.2 Proof of Correctness

We want to prove that every unsatisfied equation has an extra cost of one associated with it. At first, it would seem that this is very easy—the gadget in Fig. 1 is traversed by a path of length four if the equation is satisfied and a path of length at least five otherwise; the gadget in Fig. 5 is traversed by a path of length one if the equation is satisfied and a path of length at least two otherwise; and the gadget in Fig. 6 ensures consistency and is traversed by a tour of length six, not counting the edges that were accounted for above. Unfortunately, things are more complicated than this. Due to the consistency gadgets, the tour can leave a ticked edge when it is half-way through it, which forces us to be more careful in our analysis.

We count the number of *endpoints* that occur within the gadgets; each endpoint gives an extra cost of one half. We say that an occurrence of a literal is *traversed* if both of its connected edges are traversed, *untraversed* if none of its connecting edges are traversed, and *semitraversed* otherwise. To construct an assignment to the literals, we use the convention that a literal is true if it is either traversed or semitraversed. We need to show that there are two endpoints in gadgets that are traversed in such a way that the corresponding assignment to the literals makes the equation unsatisfied. The following lemmas are easy, but tedious, to verify by case analysis; we omit the proofs form this extended abstract:

Lemma 8. *It is locally optimal to traverse both* bridges*, i.e., both pairs of undirected edges, in the consistency gadget.*

Lemma 9. *Every semitraversed occurrence introduces at least one endpoint.*

Lemma 10. *It is always possible to change a semitraversed occurrence into a traversed one without introducing any endpoints in the consistency gadget.*

Lemma 11. *A "satisfying traversal" of the gadget in Fig. 5 has length 1, all other locally optimal traversals have length at least 2, i.e., contain at least two endpoints within the gadget.*

Lemma 12. *A "satisfying traversal" of the gadget in Fig. 7 has length 4, all other locally optimal traversals have length at least 5, i.e., contain at least two endpoints within the gadget.*

We also need to prove that the gadget we use for consistency actually implements consistency.

Lemma 13. *The gadget in Fig. 6 ensures consistency and is traversed by a tour of length 6, not counting the edges or endpoints that were accounted for in the above lemmas.*

By combining the above lemmas, we have shown the following connection between the length of an optimum tour and the number of unsatisfied equations in the corresponding instance of Hybrid.

Theorem 5. *Suppose that we are given an instance of Hybrid with n variables, m_2 equations of the form $x + \bar{y} = 1 \bmod 2$, and m_3 equations of the form $x + y + z = 0 \bmod 2$ or $x + y + \bar{z} = 0 \bmod 2$ such that:*

1. *Each variable occurs exactly three times.*
2. *There is at least one positive and at least one negative occurrence of each variable.*

Then we can construct an instance of (1,2)-ATSP with the property that a tour of length $6n + m_2 + 4m_3 + u$ corresponds to an assignment satisfying all but u of the equations in the Hybrid instance.

Corollary 2. *For any constant $\epsilon > 0$, it is **NP**-hard to approximate (1,2)-ATSP within $321/320 - \epsilon$.*

Proof. We connect Theorem 5 with Corollary 1 and obtain an instance of (1,2)-ATSP with the property that a tour of length $6 \cdot 42\nu + 60\nu + 4 \cdot 2\nu + u = 320\nu + u$ corresponds to an assignment satisfying all but u of the equations in the Hybrid instance. Since, for any constant $\epsilon' > 0$, it is **NP**-hard to distinguish the cases $u \leq \epsilon'$ and $u \geq 1 - \epsilon'$, it is **NP**-hard to approximate (1,2)-ATSP within $321/320 - \epsilon$ for any constant $\epsilon > 0$.

4 Conclusions

It should be possible to improve the reduction by eliminating the vertices that connect the equation gadgets for $x+y+z = \{0,1\}$ with each other. This reduces the cost of those equation gadgets by one, which improves our bounds—but only by a miniscule amount. The big bottleneck, especially in the (1,2) case, is the consistency gadgets. If, for the asymmetric case, we were able to decrease the cost of them to four instead of six, we would improve the bound to $237/236 - \epsilon$; if we could decrease the cost to three, the bound would become $195/194 - \epsilon$. We conjecture that some improvement for the (1,2) case is still possible along these lines.

Acknowledgments. We thank Santosh Vempala for many clarifying discussions on the subject of this paper.

References

1. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.
2. P. Berman and M. Karpinski. On some tighter inapproximability results. In *Proc. 26th ICALP*, vol. 1644 of *LNCS*, pp 200–209, 1999.
3. H.-J. Böckenhauer, J. Hromkovič, R. Klasing, S. Seibert, and W. Unger. An improved lower bound on the approximability of metric TSP and approximation algorithms for the TSP with sharpened triangle inequality. In *Proc. 17th STACS*, vol. 1770 of *LNCS*, pp 382–391, 2000.
4. H.-J. Böckenhauer and S. Seibert. Improved lower bounds on the approximability of the traveling salesman problem. *RAIRO Theoretical Informatics and Applications*, 34(3):213–255, 2000.
5. N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report CS-93-13, GSIA, Carnegie Mellon University, 1976.
6. L. Engebretsen. An explicit lower bound for TSP with distances one and two. In *Proc. 16th STACS*, vol. 1563 of *LNCS*, pp 373–382, 1999.
7. A. Frieze, G. Galbiati, and F. Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12(1):23–39, 1982.
8. J. Håstad. Some optimal inapproximability results. In *Proc. 29th STOC*, pp 1–10, 1997.
9. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pp 85–103. Plenum Press, New York, 1972.
10. C. H. Papadimitriou and S. Vempala. On the approximability of the traveling salesman problem. In *Proc. 32nd STOC*, pp 126–133, 2000.
11. C. H. Papadimitriou and S. Vempala. On the approximability of the traveling salesman problem. Manuscript, 2001.
12. C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Math. of Oper. Res.*, 18(1):1–11, 1993.
13. L. Trevisan. When Hamming meets Euclid: The approximability of geometric TSP and MST. In *Proc. 29th STOC*, pp 21–29, 1997.
14. S. Vishwanathan. An approximation algorithm for the asymmetric travelling salesman problem with distances one and two. *Inf. Process. Lett.*, 44(6):297–302, 1992.

The RPR^2 Rounding Technique for Semidefinite Programs

Uriel Feige and Michael Langberg

Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, {feige,mikel}@wisdom.weizmann.ac.il.

Abstract. Several combinatorial optimization problems can be approximated using algorithms based on semidefinite programming. In many of these algorithms a semidefinite relaxation of the underlying problem is solved yielding an optimal vector configuration $v_1 \dots v_n$. This vector configuration is then *rounded* into a $\{0, 1\}$ solution. We present a procedure called RPR^2 (Random Projection followed by Randomized Rounding) for rounding the solution of such semidefinite programs. We show that the *random hyperplane* rounding technique introduced by Goemans and Williamson, and its variant that involves *outward rotation* are both special cases of RPR^2. We illustrate the use of RPR^2 by presenting two applications. For Max-Bisection we improve the approximation ratio. For Max-Cut, we improve the tradeoff curve (presented by Zwick) that relates the approximation ratio to the size of the maximum cut in a graph.

1 Introduction

For NP-hard maximization problems, we are interested in polynomial time approximation algorithms that for every instance produce solutions whose value is guaranteed to be within a ratio of at least α from the value of the optimal solution. The parameter $0 \leq \alpha \leq 1$ is known as the *approximation ratio* of the algorithm, and the larger α is, the better.

A common method for obtaining an approximation algorithm for a combinatorial optimization problem is based on linear programming:

1. Formulate the problem as an integer linear program.
2. Relax the problem to a linear program.
3. Solve the relaxation in polynomial time, obtaining a fractional solution $x_1, \dots, x_n$.
4. Round the fractional solution to a 0/1 solution.

There are several approaches of how to round a fractional solution $x_1 \dots x_n$, and the approach to choose depends on the problem. Two common approaches are: *(a)* **Threshold rounding** in which a threshold t is set and each variable x_i is rounded to 1 if $x_i \geq t$, and to 0 otherwise (this approach is used for the Vertex Cover problem in [Hoc82]). *(b)* **Randomized rounding** in which a (monotone)

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 213–224, 2001.

rounding function $f : R \to [0,1]$ is chosen. Each variable x_i is rounded independently to 1 with probability $f(x_i)$ and to 0 otherwise (randomized rounding was introduced in [RT87], and used for example in the approximation of the Max-SAT problem [GW94]). When the rounding function f is a threshold function (0 below the threshold and 1 above it), we get threshold rounding as a special case of randomized rounding.

Goemans and Williamson [GW95] successfully extended this approach to semidefinite programming. They use a *random hyperplane* rounding technique. In our presentation below we break this rounding technique into two steps (steps 4 and 5).

1. Formulate the problem as an integer quadratic program.
2. Relax the problem to a semidefinite program.
3. Solve the relaxation in polynomial time, obtaining a vector solution $v_1 \dots v_n$.
4. Project the vector solution on a random line through the origin, obtaining a fractional solution $x_1 \dots x_n$. The value x_i is defined to be the (directed) distance of the projection of the vector v_i form the origin.
5. Round the fractional solution $x_1 \dots x_n$ to a 0/1 solution using threshold rounding. (The threshold chosen by Goemans and Williamson is 0, rounding vectors with positive projection to 1 and negative projection to 0.)

Hence both the linear programming approach and the semidefinite programming approach eventually round a fractional solution to a 0/1 solution. The main difference in this respect is that for semidefinite programming threshold rounding has always been used at this stage, whereas for linear programming it has often been the case that randomized rounding is preferred.

In this paper we study the use of a randomized rounding procedure instead of threshold rounding for the last step of the semidefinite programming approach. We call this rounding technique RPR^2 (*random projection, randomized rounding*) for short. The main contribution of this paper is in adding RPR^2 to the "tool kit" of rounding techniques for semidefinite programs. To achieve this, we do several things: *(a)* We show how the tool can be used. This includes methods for choosing the rounding function f, and methods for analyzing (or lower bounding) the resulting approximation ratio. *(b)* We identify classes of optimization problems for which RPR^2 has potential of improving the known approximation ratios (or possibly even achieving approximation ratios that match the integrality gap of the semidefinite program). *(c)* We illustrate the usefulness of RPR^2 by improving the known approximation ratios for some of these problems.

We now go on to discuss the types of problems for which RPR^2 may be useful. For simplicity and concreteness, we shall concentrate on variants of the Max-Cut problem. Given a graph $G = (V, E)$ the Max-Cut problem is the problem of finding a maximum cut of G (*i.e.* a partition of the vertex set V into two sets $(U, V \setminus U)$ that maximizes the number of edges with one end-point in U and the other in $V \setminus U$). Goemans and Williamson [GW95] use the method described above (with semidefinite programming and threshold rounding) to obtain a partition $(U, V \setminus U)$ of value at least $\alpha \simeq 0.87856$ times the value of the maximum cut in G. Recently in [FS01] it was shown that this approximation

ratio matches the integrality ratio (for the particular semidefinite program used by Goemans and Williamson), and hence we shall not try to improve it (at least not in this paper). Instead we shall consider special cases of Max-Cut for which threshold rounding (that is, the random hyperplane) produces a solution that is clearly not optimal. A simple sufficient condition for a solution not to be optimal is that of local optimality. Call a vertex *misplaced* if most of its neighbors lie in the same side of the cut. A solution is *locally optimal* if it does not have misplaced vertices. Clearly, a solution that is not locally optimal can be improved by having misplaced vertices change sides. For some instances of Max-Cut, it can be shown that if the approximation ratio of the Goemans and Williamson algorithm is indeed as bad as $\alpha \simeq 0.87856$, then necessarily the solution produced has (a substantial number of) misplaced vertices. Hence the approximation ratio can be improved by adding a local correction step to the algorithm that moves vertices from side to side until the solution becomes locally optimal. The questions that remain are how to best guide the local correction step in its choices, and how to analyze the effect of the local correction step. In some cases, RPR^2 answers both questions simultaneously.

Consider *light* Max-Cut, the problem of Max-Cut on instances where the maximum cut in a graph is not very large (below a 0.844 fraction of the edges in the graph). For this case Zwick [Zwi99] showed how to obtain approximation ratios above $\alpha \simeq 0.87856$, using the tool of *outward rotation*. We propose to use RPR^2 instead. We observe that for instances of light Max-Cut for which threshold rounding is at its worse, there are misplaced vertices. (This may not be obvious to the reader at this point, but is true nevertheless.) Hence necessarily the approximation ratio can be improved, and the only question is by how much. By a suitable choice of a rounding function f, we can use the RPR^2 technique to give solutions that are locally optimal, and moreover, we can lower bound the approximation ratio that is obtained. As we shall see, this approximation ratio is better than the one obtained by Zwick using outward rotations. This is no coincidence, because as we shall show, outward rotation can be viewed as a special case of RPR^2, but with a choice of rounding function f that produces a solution that is not locally optimal.

The use of RPR^2 above can be viewed as using a threshold scheme (random hyperplane) followed by a randomized local correction step (in the randomized rounding phase vertices change sides with probabilities related to their distance from the hyperplane). Hence the choice of rounding function f guides the local correction step, and the RPR^2 methodology gives us a way of quantifying the effect of using a local correction step. Moreover, it is straightforward to derandomize the local correction step (using the method of conditional probabilities) giving a local correction step that is deterministic.

It is fair to remark that not in all cases it is advantageous to use the RPR^2 approach in order to guide and analyze local corrections. For example, local corrections were successfully used in [FKL00] to improve the approximation ratio for Max-Cut for graphs of bounded degree. We do not think that the analysis of [FKL00] can be cast in the terminology of RPR^2.

Another class of problems for which RPR^2 may be useful is cases where threshold rounding produces infeasible solutions. Given a graph $G = (V, E)$ the Max-Bisection problem is the problem of finding a partition $(U, V \setminus U)$ of V into two equally sized sets (*i.e.* a *bisection* of V) that maximizes the number of edges cut by the partition. The algorithm of [GW95] (described above) for the Max-Cut problem on G will probably yield a partition $(U, V \setminus U)$ which is not a bisection (*i.e.* $|U| \neq |V \setminus U|$). Hence, in order to obtain a feasible partition of V (*i.e.* a bisection), the partition $(U, V \setminus U)$ must be modified (e.g., by moving vertices from the large side of the partition to the smaller one until both sides are equal). It is very difficult to analyze the effect of this additional step. There has been a sequence of papers [FJ97,Ye99,HZ00], each improving the bounds of the previous papers. We observe that RPR^2 is a natural rounding technique to use in this context, because by an appropriate choice of the rounding function f (possibly, based on the outcome of the random projection), we can guarantee that the two sides are of (nearly) the same size. We show a particular choice of rounding function f that modestly improves the known approximation ratio for Max-Bisection. We suspect that there are choices of f that give more dramatic improvements, though we are still struggling with their analysis.

A function f is called *s-linear* if for some $s \geq 0$ it is of the form $f(x) = 0$ for $x \leq -s$, $f(x) = 1$ for $x \geq s$, and $f(x) = \frac{1}{2} + \frac{x}{2s}$ for $-s \leq x \leq s$. As concrete examples of our results, we have the following theorem:

Theorem 1. *Using RPR^2 with an s-linear rounding function f, one obtains the following approximation ratios.*

- *For light Max-Cut for instances in which the optimal cut contains at most* 0.6 *of the edges, the ratio is at least 0.9128. (Previous best bound was below 0.9119.)*
- *For Max-Bisection the ratio is at least 0.7027. (Previous best bound was below 0.7017.)*

Our paper is structured as follows. In Section 2 we review the random hyperplane and outward rotation rounding techniques. In Section 3 we define RPR^2 and show that outward rotation is a special case of RPR^2. In Section 4 we analyze the use of RPR^2 on the Max-Cut and Max-Bisection problems. Finally, in Section 5 we offer some concluding remarks. Due to space limitations, the results of our work are presented without detailed proof. In most cases, [FL01] (the extended version of our work) contains the missing details.

2 SDP Relaxation of Max-Cut and Various Roundings

Consider the Max-Cut problem on a graph $G = (V, E)$ with $|V| = n$. It can be represented as a quadratic integer program:

(QI-MC)	Maximize	$\sum_{e_{ij} \in E} \frac{1 - x_i \cdot x_j}{2}$	
	subject to:		
		$x_i \in \{-1, 1\}$	for $1 \leq i \leq n$

The above program can be understood as follows. With each vertex $i \in V$ we associate a variable x_i, and the value of x_i (which is either $+1$ or -1) indicates in which side of the cut the respective vertex is placed. For each edge $e_{ij} \in E$, if $x_i \neq x_j$ (corresponding to the case in which e_{ij} is cut) then the value of $\frac{1-x_i \cdot x_j}{2}$ is 1, and if $x_i = x_j$ then this value is 0.

The requirement $x_i \in \{-1, 1\}$ can be relaxed by representing each variable x_i by a unit n-dimensional vector $v_i \in S^n$ (here S^n is the unit sphere) and the multiplication $x_i \cdot x_j$ by the inner product $\langle v_i, v_j \rangle$.

(SDP-MC)	Maximize	$\sum_{e_{ij} \in E} \frac{1-\langle v_i, v_j \rangle}{2}$	
	subject to:		
		$v_i \in S^n$	for $1 \leq i \leq \lvert V \rvert$

As every solution of *(QI-MC)* is also a solution of *(SDP-MC)*, the value of *(SDP-MC)* is at least as large as that of *(QI-MC)*. *(SDP-MC)* can be solved (up to arbitrary precision) in polynomial time using semidefinite programming (see [GW95]). A solution to *(SDP-MC)* is a set of unit vectors in R^n, rather than a cut of G. To obtain a cut $(U, V \setminus U)$ of G we *round* the set of optimal vectors $v_1 \ldots v_n$ obtained by solving *(SDP-MC)*. One such rounding technique, presented by Goemans and Williamson [GW95] is the *random hyperplane* rounding technique.

Let $r = r_1 \ldots r_n$ be a random variable with an n dimensional standard normal distribution (*i.e.* each coordinate r_i is an independent random variable with standard normal distribution). It can be seen that r is spherically symmetric, namely the direction specified by the vector $r \in R^n$ is uniformly distributed (see for instance [Ren70]). In the random hyperplane rounding technique a random vector r of the above distribution is chosen and the vectors $v_1 \ldots v_n$ are partitioned into two sets according to the sign of the inner product $\langle v_i, r \rangle$. That is, a cut $(U, V \setminus U)$ is defined by the set $U = \{i \mid \langle v_i, r \rangle > 0\}$.

Using the semidefinite program *(SDP-MC)* and the random hyperplane rounding technique, [GW95] obtain a 0.87856 approximation ratio for the Max-Cut problem. A number of other approximation algorithms for various problems have been designed using semidefinite programming and variations of the random hyperplane rounding technique (for example [FG95,KMS98,FJ97,KZ97,Zwi99], [Ye99,FKL00]). In some of these algorithms, the vectors $v_1 \ldots v_n$ obtained by solving a semidefinite relaxation are rearranged prior to the use of random hyperplane rounding.

One method used to rearrange the vectors $v_1 \ldots v_n$ is *outward rotations* [Nes98,Ye99,Zwi99]. Let $\gamma \in [0, 1]$. Given a set of vectors $v_1 \ldots v_n$ in R^n, obtained by the solution of a semidefinite program, the γ-outward rotation of $v_1 \ldots v_n$ are a set of new vectors $\hat{v}_1 \ldots \hat{v}_n$ in R^{2n}. The vector $\hat{v}_i$ is defined to be $\sqrt{1-\gamma} v_i + \sqrt{\gamma} e_i \in R^{2n}$ where the original vectors v_i are viewed as vectors in R^{2n} (R^n being a subspace of R^{2n}) and $e_1 \ldots e_n$ are a set of orthonormal vectors in R^{2n} that are also orthogonal to the vectors $v_1 \ldots v_n$. In general, when $\gamma = 1$ the γ-outward rotation of the vectors $v_1 \ldots v_n$ is a new vector configuration in which all vectors are orthogonal, and when $\gamma = 0$ the γ-outward rotation does not change the vectors $v_1 \ldots v_n$. For intermediate γ, the γ-outward rotation is

somewhere in between. Outward rotation has been used in the design of approximation algorithms for special instances of the Max-Cut problem, and other problems closely related to Max-Cut (for example [Zwi99,Ye99,HZ00]).

3 *Random Projection, Randomized Rounding* (RPR^2)

Let $v_1 \dots v_n$ be a set of vectors obtained by the solution of a semidefinite relaxation. We define a family of rounding procedures parameterized by a function $f : R \to [0,1]$. We denote this family of rounding procedures as the *random projection, randomized rounding* (RPR^2) family. An RPR^2 procedure using f has two steps and is defined as:

Step 1 (Projection): Project the vectors $v_1 \dots v_n$ onto a random one dimensional subspace (*i.e.* a line). This is done by choosing a random variable r with n dimensional standard normal distribution, and projecting each v_i onto the one dimensional subspace containing r. For each i, let x_i be the directed distance (times $\|r\|$) of the projected vector v_i from the origin (*i.e.* $x_i = \langle v_i, r \rangle$).

Step 2 (Randomized rounding): Define the $\{0,1\}$ solution $a_1 \dots a_n$: for each i set a_i to be 1 independently with probability $f(x_i)$.

The standard random hyperplane rounding technique presented in [GW95] in the approximation of the Max-Cut problem is a member of the RPR^2 family. The function f corresponding to random hyperplane rounding is the function which is 1 for all $x > 0$, and zero otherwise. Later in this section we show that the outward rotation rounding technique is also a special case of RPR^2. In Section 4 we study the use of RPR^2 on the "light" Max-Cut problem and on the Max-Bisection problem. For both these problems outward rotation was used in order to obtain the previously best approximation ratios. We show functions f which when used in RPR^2 give better approximation ratios.

Analyzing RPR^2: Let $v_1 \dots v_n$ be the solution of a semidefinite relaxation on a given graph $G = (V, E)$, and let $a_1 \dots a_n$ be the $\{0,1\}$ solution obtained by using RPR^2 with some function f. An edge e_{ij} is cut if $a_i \neq a_j$. As RPR^2 is a randomized procedure, the number of edges cut is a random variable. We wish to compute its expectation. For this, we analyze the probability of the event "$a_i \neq a_j$".

Let $r = r_1 \dots r_n$ be an n dimensional standard normal vector. In general, given r, the probability that "$a_i \neq a_j$" depends on the vectors v_i, v_j, and the function f. Hence, integrating over all possible r one can compute the probability of this event. However, as r is spherically symmetric this probability can be computed using two independent standard normal random variables r_1, r_2, and the angle between v_i and v_j alone. Given two vectors v_i and v_j that form an angle of θ_{ij}, let $P_f(\theta_{ij})$ denote the probability that the corresponding values a_i and a_j differ. Let $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ be the density function of a standard normal random variable. The following lemma is straightforward (details appear in [FL01]).

Lemma 1. *Let* $\theta \in [0,\pi]$, $z(r_1,r_2) = \cos(\theta)r_1 + \sin(\theta)r_2$.

$$P_f(\theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} [f(r_1)(1-f(z(r_1,r_2))) + f(z(r_1,r_2))(1-f(r_1))]\phi(r_1)\phi(r_2)dr_1dr_2$$

By linearity of expectation, the expected number of edges cut is $\sum_{e_{ij}\in E} P_f(\theta_{ij})$, where θ_{ij} is the angle formed by the vectors v_i and v_j corresponding to e_{ij}. Dividing this value by $|E|$ we get the expected *fraction* of edges cut, which we denote by $E[Cut_f]$.

For an edge that makes an angle of θ, let $SDP(\theta) = (1-\cos\theta)/2$ be its contribution to the semidefinite program. Then a convenient lower bound on the approximation ratio achieved for Max-Cut by RPR^2 with a rounding function f is $\min_{\theta>0} P_f(\theta)/SDP(\theta)$. For light Max-Cut, this lower bound is too pessimistic. The angle θ minimizing the above expression cannot hold simultaneously for all edges, because then the graph would contain a cut that is too large. A stronger lower bound on the approximation ratio can be derived by more detailed analysis, following principles outlined in [Zwi99]. For Max-Bisection, there are more complications, because the cut obtained by the rounding technique is not necessarily a bisection. An additional step of moving vertices from the larger side to the smaller one is used, and analyzing its effect (or at least, providing lower bounds), can be done using the techniques outlined in [FJ97,Ye99,HZ00]. The numerical bounds for the approximation ratios that are presented in this paper were derived by using analysis that follows the principles developed in [FJ97, Zwi99,Ye99,HZ00]).

Outward rotation is a special case of RPR^2: Let $v_1 \dots v_n$ be as above. Let γ be some value in $[0,1]$. Recall that the γ-outward rotation $\hat{v}_1 \dots \hat{v}_n$ of the vectors $v_1 \dots v_n$ is defined by $\hat{v}_i = \sqrt{1-\gamma}v_i + \sqrt{\gamma}e_i \in R^{2n}$.

In the standard use of outward rotation, a $\{0,1\}$ solution $a_1 \dots a_n$ is obtained by rounding the vectors $\hat{v}_1 \dots \hat{v}_n$ by a random hyperplane. Specifically let $r = r_1 \dots r_{2n}$ be a random vector with a $2n$-dimensional standard normal distribution. Define the solution $a_1 \dots a_n$ by setting a_i to be one iff the inner product $\langle \hat{v}_i, r\rangle$ is positive. It is convenient to describe the solution $a_1 \dots a_n$ as the subset $U = \{i \in V \mid a_i = 1\}$ of V, *i.e.* $U = \{i \in V \mid \langle \hat{v}_i, r\rangle > 0\}$. Using the definition of $\hat{v}_i$ and the spherical symmetry of r, we have that the set U obtained is equal to $\{i \in V \mid \sqrt{1-\gamma}\langle v, r_1 \dots r_n\rangle + \sqrt{\gamma}r_{n+i} > 0\}$.

We would like to obtain the exact set U without the use of outward rotations. Instead we would like to use RPR^2. Let $\phi(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}$ and $\Phi(x) = \int_{-\infty}^{x}\phi(x)dx$ be the density function and distribution function of a standard normal random variable. We obtain the following theorem (detailed proof can be found in [FL01]).

Theorem 2. *For any* $\gamma \in [0,1]$, *let* $f_\gamma = \Phi\left(x\sqrt{\frac{1-\gamma}{\gamma}}\right)$. *Using* RPR^2 *with* f_γ *is equivalent to* γ*-outward rotation followed by random hyperplane rounding.*

In cases where outward rotation is used, it is natural to ask whether RPR^2 in combination with a different rounding function f can give better approximation

ratios. It is our belief that the answer to this question is in general positive. That is, whenever outward rotation gives better approximation ratios than the random hyperplane rounding technique, one should expect RPR^2 to offer further improvements.

Let us note that for RPR^2, if we restrict ourselves to *nice* behaving rounding functions, finding the optimal rounding function f is not really a problem. For any fixed $\epsilon > 0$, there is a constant number of functions f (where this constant depends exponentially on $1/\epsilon$) such that at least one of them has expected approximation ratio within an additive error of at most ϵ from the optimal f. (This can be shown using concepts such as ϵ-nets.) An RPR^2 algorithm can even try out all these functions at run time and take the best result obtained. Hence, we may always assume that RPR^2 is performed with the best possible rounding function. The problem is in analyzing the approximation ratio that one obtains. Here it is useful to select one particular easy to analyze rounding function f, to compute the expected approximation ratio that this f gives, and use it as a lower bound on the approximation ratio of the RPR^2 scheme. In this respect, outward rotations are helpful, as they can be analyzed not only using the integrals of Lemma 1, but also via other techniques (as in [Ye99,Zwi99]), and these other techniques are often simpler to use. It would be fair to say that previous work on outward rotation served as inspiration to much of the work reported in the current paper.

4 Applications of RPR^2

Light Max-Cut:

Let $G = (V, E)$ be a given graph, let $v_1 \dots v_n$ be the optimal vector configuration obtained by solving the semidefinite relaxation *(SDP-MC)* presented in [GW95] (and in Section 2) of G, and let Z be the value of this relaxation. Can RPR^2 (with some specific f) be used on the Max-Cut problem in order to improve the approximation ratio of $\alpha \simeq 0.87856$ proved in [GW95] ? If we define α to be the ratio between the expected value of the cut obtained using RPR^2, and the value of the semidefinite relaxation Z, the answer is negative. This is due to a recent work of Feige and Schechtman [FS01] that shows that the integrality gap of this relaxation is arbitrarily close to $1/\alpha$. Therefore, we will not try to improve the approximation ratio α on general instances G. Instead we shall consider special cases of Max-Cut.

Consider parameterizing the instances of Max-Cut according to the ratio between the value of the semidefinite relaxation Z and the total number of edges W. Goemans and Williamson [GW95] study Max-Cut restricted to instances G for which this ratio is greater that 0.844. For each value $t \in (0.844, 1]$, they show that using standard random hyperplane rounding on instances G for which $Z = tW$, will yield a cut of value at least $\alpha_t Z$ where $\alpha_t > \alpha$ (this implies an approximation ratio of α_t). In [FS01] it is shown that the integrality gap of *(SDP-MC)* on these restricted instances is arbitrarily close to $1/\alpha_t$. Therefore,

we will not try to improve the algorithm of [GW95] restricted on such instances either.

Zwick [Zwi99] studies Max-Cut restricted to instances G for which $Z = tW$ and $t < 0.844$. We call this "light" Max-Cut. For these instances Zwick shows how to obtain approximation ratios $\alpha_t > \alpha$, using outward rotation followed by random hyperplane rounding. The value of the integrality gap *(SDP-MC)* on these restricted instances is not clear. We analyze the use of RPR^2 on light Max-Cut. Roughly speaking, we show *(a)* Necessary conditions for a rounding function f to be the one that maximizes the expected value of the cut obtained by RPR^2. *(b)* Outward rotation (a special case of RPR^2) is not the best rounding function for RPR^2, as it does not satisfy these necessary conditions. *(c)* We present an s-linear rounding function that gives an approximation ratio strictly above the ratio of α_t presented in [Zwi99]. We suspect that RPR^2 (with the optimal choice of f) achieves an approximation ratio that matches the inverse of the integrality gap of *(SDP-MC)* (as a function of t). We are trying to extend the techniques of [FS01] in order to prove this.

Our analysis involves the numerical evaluation of double integrals (presented in Lemma 1). These evaluations have been performed using MATLAB functions within precision of 10^{-6}. As such computations are time consuming, item *(b)* and *(c)* above are shown for a few values of t (namely $t = 0.55, 0.6, 0.7$).

Properties of the best function f for RPR^2: Given a graph $G = (V, E)$ and a set of vectors $v_1 \dots v_n$ obtained by solving *(SDP-MC)* on G, let $E[Cut_f]$ be the expected fraction of edges cut by using RPR^2 with a function f. Call a function *well behaved* if it is piecewise continuous. We are interested in finding a well behaved function f^* that maximizes $E[Cut_f]$.

For light Max-Cut we identify a necessary condition for any well behaved function f that maximizes $E[Cut_f]$. We use this necessary condition to prove that the functions f corresponding to outward rotation are not optimal. Moreover, this necessary condition helps to guide us in finding a better rounding function (in our case, an s-linear function), without resorting to a tedious exhaustive search type approach for such a function (as mentioned in Section 3).

A natural property that one could expect from an optimal function f^* is that rounding the vectors $v_1 \dots v_n$ using RPR^2 with f^* yields a cut $(U, V \setminus U)$ which is expected to be locally optimal (*i.e.* there are no vertices with an expected majority of neighbors on their side of the cut). For instance for the function $f(x) \equiv 1/2$ this property holds. The necessary condition we suggest is closely related to this "local optimality" property.

Let $G = (V, E)$ be a given graph and $v_1 \dots v_n$ be the set of vectors obtained by solving *(SDP-MC)* on G. Let f be some RPR^2 function. Recall that $P_f(\theta)$ (defined in Lemma 1) measures the probability that using RPR^2 with the function f, two vectors v_i and v_j that form an angle of θ have corresponding values a_i and a_j that differ. Let $E[Cut_f]$ be the expected fraction of edges cut by using RPR^2 with f. That is, $E[Cut_f]$ is a normalized sum of $P_f(\theta_{ij})$, where θ_{ij} is the angle between the vectors corresponding to edges e_{ij} in E. Consider the probability $P_f(\theta_{ij})$ conditioned on the event that the inner product between v_i

and the random vector r used in RPR^2 is fixed to be a specific value r_1. We denote this probability as $P_f(\theta_{ij})|r_1$. Define $E[Cut_f \mid r_1]$ as the corresponding normalized sum of $P_f(\theta_{ij})|r_1$.

Theorem 3. *If f^* is an optimal (well behaved) RPR^2 function then for all r_1 we have that $E[Cut_{f^*} \mid r_1] \geq 1/2$ with equality if $0 < f(r_1) < 1$ (i.e. $f(r_1) \notin \{0, 1\}$).*

Proof of the above theorem appears in [FL01]. We would like to note that our proof is done in a constructive manner. That is, if f is some RPR^2 function that does not satisfy the above conditions in some interval Δ, we show how to modify f in Δ to obtain a new function f^* such that $E[Cut_{f^*}] > E[Cut_f]$, thus implying that f is not optimal.

Outward rotation is not the best function for RPR^2: For an instance $G = (V, E)$ of Max-Cut, let $W = |E|$, let $v_1 \dots v_n$ be the vector configuration obtained by solving relaxation *(SDP-MC)* on G and let Z be the value of the relaxation. Let $t \in [0.5, 0.844)$. Assume a graph $G = (V, E)$, with a corresponding vector configuration $v_1 \dots v_n$ of value $Z = tW$. In [Zwi99] it is shown that rounding $v_1 \dots v_n$ using γ-outward rotation, an expected approximation ratio strictly above α_t will be obtained unless for each edge e_{ij} in E, the corresponding vectors v_i and v_j form an angle of either zero or θ_t (where θ_t is some specific angle greater than $\pi/2$ that depends on t).

In other words, only on graphs $G = (V, E)$ with a corresponding vector configuration in which a δ fraction of edges e_{ij} in E have corresponding vectors v_i and v_j that form an angle of zero, and a $1-\delta$ fraction of edges have corresponding vectors that form an angle of θ_t, does the algorithm of [Zwi99] obtain an approximation ratio of no better than α_t. On all other graphs the algorithm of [Zwi99] has an approximation ratio strictly greater than α_t.

Let f_γ be the RPR^2 function corresponding to γ-outward rotation. It can be seen ([FL01]) that for such worst case graphs there exists a non negligible interval $\Delta \subseteq R$, and a constant $\varepsilon > 0$ such that $E[Cut_{f_\gamma} \mid r_1] < 1/2 - \varepsilon$ for all $r_1 \in \Delta$ (for instance for $t = 0.6$ we have $E[Cut_{f_\gamma} \mid r_1] < 0.493$ for $r_1 \in [0.3, 0.4]$). By a quantitative version of Theorem 3 we may construct a new function f^* by modifying f_γ in the interval Δ such that $E[Cut_{f^*}] > E[Cut_{f_\gamma}] + poly(\varepsilon)$. We conclude

Theorem 4. *There exists a constant $\varepsilon > 0$, such that using RPR^2 on the worst case graphs of [Zwi99] an approximation ratio of $\alpha_t + \varepsilon$ can be obtained.*

This implies an improved approximation algorithm for Max-Cut on general instances G with $Z = tW$. If the given graph has a vector configuration *close* to the worst case configuration use the best function for RPR^2, otherwise use the original algorithm of [Zwi99] (we rely on the fact that RPR^2 rounding is continuous with respect to the vector configuration $v_1 \dots v_n$).

As noted previously, our analysis involves the numerical evaluation of integrals, thus the above theorem has been proven for $t = 0.55, 0.6, 0.7$. We have no reason to believe that our results depend on these particular values of t.

An example for superior RPR^2 functions: We have shown that an approximation ratio greater than α_t can be obtained on graphs G with $Z = tW$

by improving the approximation ratio obtained on worst case graphs of [Zwi99]. Following we show that such an improvement can be proven directly. That is, given a value t and a graph G with $Z = tW$, we are interested in proving a lower bound on the expected value of the cut obtained by RPR^2. This can be done by choosing some function f, and analyzing the value of $P_f(\theta)$ for every $\theta \in [0, \pi]$.

Let s be some threshold, recall that an s-linear function f_s^* is the continuous function that is zero for all $x \leq -s$, one for all $x \geq s$, and linear for all $x \in (-s, s)$. By replacing the function f_{γ_t} that corresponds to outward rotation, by an s-linear function f_s^* the following approximation ratios were achieved. For $t = 0.55$ and $s = 0.96$ a ratio of 0.942562 (as opposed to 0.941282 of [Zwi99]), for $t = 0.6$ and $s = 0.635$ a ratio of 0.912809 (as opposed to 0.911890 of [Zwi99]), and for $t = 0.7$ and $s = 0.263$ a ratio of 0.886453 (as opposed to 0.886251 of [Zwi99]). The functions f_s^* are not claimed to be optimal, but as we have checked many different functions, we believe that they are close to being so. Hence, it seems that the original functions corresponding to outward rotation are very close to being optimal.

Max-Bisection:

Given a graph $G = (V, E)$ the Max-Bisection problem is the problem of finding a partition $(U, V \setminus U)$ of V into two equally sized sets (*i.e.* a *bisection* of V) that maximizes the number of edges cut by the partition. A number of approximation algorithms for Max-Bisection based on semidefinite programming have been suggested [FJ97,Ye99,HZ00], yielding approximation ratios of 0.6514, 0.699, 0.7016 respectively. In these algorithms, a semidefinite relaxation of Max-Bisection is solved yielding a set of vectors $v_1 \ldots v_n$. These vectors are then rounded (using the random hyperplane or outward rotation technique) in order to obtain a cut $(U, V \setminus U)$ of G. This cut is not necessarily a bisection, thus the cut $(U, V \setminus U)$ is modified by moving vertices from the large side of the cut to the smaller side until both sides are equal. As in the case of Max-Cut, we analyze the use of RPR^2 in the algorithm above and conclude the following theorem (our analysis is based on that presented in [HZ00]).

Theorem 5. *Using RPR^2 with a 0.605-linear rounding function, Max-Bisection can be approximated within an approximation ratio of 0.7027.*

5 Conclusions

Many questions remain open, but seem within reach. For "light" Max Cut, we suspect that RPR^2 (with the optimal choice of f) achieves an approximation ratio that matches the integrality ratio (as a function of the relative size of the maximum cut). We are trying to extend the techniques of [FS01] in order to prove this. For Max Bisection, we suspect that more substantial improvements of the approximation ratio can be proven for other choices of rounding function f. For some other problems, especially those currently analyzed using outward rotation (such as Not-All-Equal-3SAT [Zwi99]), it is natural to assume that the approximation ratio can be improved using RPR^2, but this needs to be seen.

Acknowledgments. The first author is the Incumbent of the Joseph and Celia Reskin Career Development Chair. The second author would like to thank Naftali Langberg for helpful discussions. This research was supported in part by project RAND APX of the European Community.

References

[FG95] U. Feige and M.X. Goemans. Approximating the value of two prover proof systems with applications to Max-2-Sat and Max-Dicut. *Proc. of the 3rd Israel Symposium on Theory of Computing and Systems*, pages 182–189, 1995.

[FJ97] A. Frieze and M. Jerrum. Improved approximation algorithms for Max-k-Cut and Max-Bisection. *Algorithmica*, 18:67–81, 1997.

[FKL00] U. Feige, M. Karpinski, and M. Langberg. Improved approximation of Max-Cut on graphs of bounded degree. *ECCC*, TR00-021, 2000.

[FL01] U. Feige and M. Langberg. The RPR^2 rounding technique for semidefinite programs. *Manuscript, http://www.wisdom.weizmann.ac.il/∼mikel*, 2001.

[FS01] U. Feige and G. Schechtman. On the optimality of the random hyperplane rounding technique for MAX CUT. *Manuscript*, 2001.

[GW94] M.X. Goemans and D.P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7(4):656–666, 1994.

[GW95] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of ACM*, 42:1115–1145, 1995.

[Hoc82] D.S. Hochbaum. Approximation algorithms for the set covering and vertex cover problem. *SIAM J. of Computing*, 11(3):555–556, 1982.

[HZ00] E. Halperin and U. Zwick. Improved approximation algorithms for maximum graph bisection problems. *Manuscript*, 2000.

[KMS98] D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *Journal of ACM*, 45(2):246–265, 1998.

[KZ97] H. Karloff and U. Zwick. A 7/8-approximation algorithm for Max-3-Sat? *In Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 406–415, 1997.

[Nes98] Y. E. Nesterov. Semidefinite relaxation and nonconvex quadratic optimization. *Optimization Methods and Software*, 9:141–160, 1998.

[Ren70] A. Renyi. Probability theory. *Elsevier, New York*, 1970.

[RT87] P. Raghavan and C.D. Thompson. Randomized rounding : A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(7):365–374, 1987.

[Ye99] Y. Ye. A 0.699-approximation algorithm for Max-Bisection. *Manuscript, available at URL http://dollar.biz.uiowa.edu/col/ye/*, 1999.

[Zwi99] U. Zwick. Outward rotations: a new tool for rounding solutions of semidefinite programming relaxations, with application to Max-Cut and other problems. *In Proceedings of the 31th ACM Symposium on Theory of Computing*, pages 679–687, 1999.

Approximation Algorithms for Partial Covering Problems

Extended Abstract

Rajiv Gandhi[1], Samir Khuller[2], and Aravind Srinivasan[3]

[1] Department of Computer Science, University of Maryland, College Park, MD 20742. Research supported by NSF Award CCR-9820965.
gandhi@cs.umd.edu.

[2] Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Research supported by NSF Award CCR-9820965 and an NSF CAREER Award CCR-9501355.
samir@cs.umd.edu.

[3] Bell Labs, Lucent Technologies, 600-700 Mountain Avenue, Murray Hill, NJ 07974.
srin@research.bell-labs.com.

Abstract. We study the generalization of covering problems to *partial covering*. Here we wish to cover only a desired number of elements, rather than covering all elements as in standard covering problems. For example, in k-set cover, we wish to choose a minimum number of sets to cover at least k elements. For k-set cover, if each element occurs in at most f sets, then we derive a primal-dual f-approximation algorithm (thus implying a 2-approximation for k-vertex cover) in polynomial time. In addition to its simplicity, this algorithm has the advantage of being parallelizable. For instances where each set has cardinality at most three, we obtain an approximation of 4/3. We also present better-than-2-approximation algorithms for k-vertex cover on bounded degree graphs, and for vertex cover on expanders of bounded *average* degree. We obtain a polynomial-time approximation scheme for k-vertex cover on planar graphs, and for covering points in R^d by disks.

Keywords and Phrases: Approximation algorithms, partial covering, set cover, vertex cover, primal-dual methods, randomized rounding.

1 Introduction

Covering problems are widely studied in discrete optimization: basically, these problems involve picking a least-cost collection of sets to cover elements. Classical problems in this framework include the general set cover problem, of which a widely studied special case is the vertex cover problem. (The vertex cover problem is a special case of set cover in which the edges correspond to elements and vertices correspond to sets; in this set cover instance, each element is in exactly two sets.) Both these problems are NP-hard and polynomial-time approximation algorithms for both are well studied. For set cover see [8,23,26]. For vertex cover see [3,4,9,18,19,27].

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 225–236, 2001.

In this paper we study the generalization of "covering" to "partial covering" [24,28]. Specifically, in k-set cover, we wish to find a minimum number (or, in the weighted version, a minimum weight collection) of sets that cover at least k elements. When k is the total number of elements, we obtain the regular set cover problem; similarly for k-vertex cover. (We sometimes refer to k-set cover as "partial set cover", and k-vertex cover as "partial vertex cover"; the case where k equals the total number of elements is referred to as "full coverage".) This generalization is motivated by the fact that real data (in clustering for example) often has errors (also called outliers). Thus, discarding the (small) number of constraints posed by such errors/outliers is permissible. Suppose we need to build facilities to provide service within a fixed radius to a certain fraction of the population. We can model this as a partial set cover problem. The main issue in partial covering is: which k elements should we choose to cover? If such a choice can be made judiciously, we can then invoke a set cover algorithm. Other facility location problems have recently been studied in this context [7].

Regarding vertex cover, a very simple approximation algorithm for the unweighted case is attributed to Gavril and Yannakakis, and can be found, e.g., in [10]: take a maximal matching and pick all the matched vertices as part of the cover. The size of the matching (number of edges) is a lower bound on the optimal vertex cover, and this yields a 2-approximation. This algorithm fails for partial covering, since the lower bound relies on the fact that all the edges have to be covered: in general, approximation algorithms for vertex cover may return solutions that are much larger than the optimal value of a given k-vertex cover instance. The first approximation algorithm for k-vertex cover was given in [6]. Their 2-approximation algorithm is based on a linear programming (LP) formulation: suitably modifying and rounding the LP's optimal solution. A faster approximation algorithm achieving the same factor of 2 was given in [21]; here, the key idea is to relax the constraint limiting the number of uncovered elements and searching for the dual penalty value. More recently, a 2-approximation based on the elegant "local ratio" method was given in [5].

Problem Definitions and Previous Work

k-Set Cover: Given a set $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$, a collection $\mathcal{S}$ of subsets of $\mathcal{T}$, $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, a cost function $c : \mathcal{S} \to \mathcal{Q}^+$, and an integer k, find a minimum cost sub-collection of $\mathcal{S}$ that covers at least k elements of $\mathcal{T}$.

For the full coverage version, a $\ln n+1$ approximation was proposed in [23,26]. This analysis of the greedy algorithm can be improved to $H(\Delta)$ (see the proof in [10]) where Δ is the size of the largest set. ($H(k) \doteq \sum_{i=1}^{k} 1/i = \ln k + \Theta(1)$.) Chvátal [8] generalized this to the case when sets have costs. Slavík [29] shows the same bound for the partial cover problem. When $\Delta = 3$, Duh and Fürer [11] gave a 4/3-approximation for the full coverage version. They extended this result to get a bound of $H(\Delta) - \frac{1}{2}$ for full coverage. When an element belongs to at most f sets Hochbaum [18] gives a f-approximation.

k-Vertex Cover: Given a graph $G = (V, E)$, a cost function $c : V \to \mathcal{Q}^+$, and an integer k, find a minimum cost subset of V that covers at least k edges of G.

Several 2-approximation algorithms are known for this; see [6,21,5].

Geometric Covering Problem: Given n points in a plane, find a minimally sized set of disks of diameter D that would cover at least k points.

Previous Results: The full coverage version is well-studied. This problem is motivated by the location of emergency facilities as well as from image processing (see [20] for additional references). For the special case of geometric covering problems, a polynomial-time approximation scheme is shown in [22].

Our Results

Full proofs of the claims in this paper are given in [12].

k-Set Cover: For the special case when each element is in at most f sets, we combine a primal-dual algorithm [9,15] with a thresholding method to obtain an f-approximation. One advantage of our method, in addition to its simplicity, is that it can be easily parallelized by changing the algorithm slightly. The resulting approximation factor is $f(1+\epsilon)$, where $\epsilon > 0$ is any desired constant. The number of parallel rounds is $O(\log n)$ once we fix $\epsilon > 0$. The number of processors required in linear in the problem size. This is the first parallel approximation algorithm for any partial covering problem. For set cover where the sets have cardinality at most Δ there are results (starting from [13,16]) by Duh and Fürer [11] for set cover (full coverage) that improve the $H(\Delta)$ bound to $H(\Delta) - \frac{1}{2}$. For example, for $\Delta = 3$ they present a $\frac{4}{3}$ $(= H(3) - \frac{1}{2})$ approximation using "semi-local" optimization rather than a $\frac{11}{6}$-approximation obtained by the simple greedy algorithm. For the case $\Delta = 3$, we can obtain a $\frac{4}{3}$ bound for the partial coverage case.

k-Vertex Cover: By switching to a probabilistic approach to rounding the LP relaxation of the problem, we obtain improved results for k-vertex cover, where we wish to choose a minimum number of vertices to cover at least k edges. An outstanding open question for vertex cover (full coverage) is whether the approximation ratio of 2 is best-possible; see, e.g., [14]. Thus, it has been an issue of much interest to identify families of graphs for which *constant-factor approximations better than* 2 (which we denote by Property (P)) are possible. In the full coverage case, Property (P) is true for graphs of bounded *maximum* degree; see, e.g., [17]. How can we extend such a result? Could Property (P) hold for graphs of constant *average* degree? This is probably not the case, since this can be shown to imply Property (P) for all graphs. As a step toward seeing which graph families of constant average degree enjoy property (P), we show that for *expander* graphs of bounded average degree, Property (P) is true. We also show Property (P) for k-vertex cover in the case of bounded maximum degree and arbitrary k; this is the first Property (P) result for k-vertex cover, to our knowledge. We also present certain new results for multi-criteria versions of k-vertex cover.

Geometric Covering: There is a polynomial approximation scheme based on dynamic programming for the full coverage version [22]. For the partial coverage version since we do not know which k points to cover, we have to define a new dynamic program. This makes the implementation of the approximation scheme due to [22] more complex, although it is still a polynomial-time algorithm.

k-Vertex Cover for Planar Graphs: We are able to use the dynamic programming ideas developed for the geometric covering problem to design a polynomial-time approximation scheme (PTAS) for k-vertex cover for planar graphs. This is based on Baker's method for the full covering case [2]. The details are omitted from this extended abstract; the interested reader is referred to [12].

2 k-Set Cover

The k-Set Cover problem can be formulated as an integer program as follows. We assign a binary variable x_j for each $S_j \in \mathcal{S}$ i.e $x_j \in \{0,1\}$. In this formulation, $x_j = 1$ iff set S_j belongs to the cover. A binary variable y_i is assigned to each element $t_i \in \mathcal{T}$. $y_i = 1$ iff t_i is not covered. Clearly, there could be at most $n - k$ such uncovered elements. An LP relaxation is obtained by letting the variables be *reals* in $[0,1]$. The LP is to minimize $\sum_{j=1}^{m} c(S_j) \cdot x_j$, subject to: (i) $y_i + \sum_{j:t_i \in S_j} x_j \geq 1$, $i = 1, 2, \ldots, n$; (ii) $\sum_{i=1}^{n} y_i \leq n - k$; (iii) $x_j \geq 0$, $j = 1, 2, \ldots, m$; and (iv) $y_i \geq 0$, $i = 1, 2, \ldots, n$. The dual LP contains a variable u_i (for each element $t_i \in \mathcal{T}$) corresponding to each of the first n constraints in the above LP. The dual variable z corresponds to the $(n+1)^{th}$ constraint in the above LP formulation. The dual LP is to maximize $\sum_{i=1}^{n} u_i - (n-k) \cdot z$ subject to: (i) $\sum_{i:t_i \in S_j} u_i \leq c(S_j)$ for $j = 1, 2, \ldots, m$, (ii) $0 \leq u_i \leq z$ for $i = 1, 2, \ldots, n$, and (iii) $z \geq 0$.

The algorithm SetCover does the following. The algorithm "guesses" the set with the highest cost in the optimal solution by considering each set in turn to be the highest cost set. For each set that is chosen, to be the highest cost set, say S_j, S_j along with all the elements it contains is removed from the instance and is included as part of the cover for this guess of the highest cost set. The cost of all sets having a higher cost than $c(S_j)$ is raised to ∞. $I_j = (\mathcal{T}^j, \mathcal{S}^j, c', k_j)$ is the modified instance. SetCover then calls Primal-Dual on I_j which uses a primal dual approach [15] to return a set cover for I_j. In Primal-Dual, the dual variables u_i are increased for all $t_i \in \mathcal{T}^j$ until there exists a set S_i such that $\sum_{i:t_i \in S_i} u_i = c'(S_i)$. Sets are chosen this way until the cover is feasible. The algorithm then chooses the minimum cost solution among the m solutions found. The pseudo-code for this algorithm can be found in [12].

Theorem 1. SetCover$(\mathcal{T}, \mathcal{S}, c, k)$ *returns a f-approximate solution, where f is the highest frequency of any element i.e. an element appears in at most f sets.*

Corollary 1. SetCover(E, V, c, k) *gives a 2-approximate solution for k-Vertex Cover.*

2.1 Parallel Implementation of Partial Set Cover Algorithm

We assume as before that each element belongs to at most f sets. The framework for the algorithm is the same as the one we described for the primal-dual serial algorithm. The parallel algorithm runs in "rounds". In each round,

we simultaneously raise all dual variables u_i corresponding to the uncovered elements. In the serial algorithm we pick one set in each iteration, namely a set S_j such that $(\sum_{i:t_i \in S_j} u_i = c'(S_j))$. (Recall that c' denotes the modified cost function.) We change this step in the algorithm to pick *all* sets such that $(c'(S_j) - \sum_{i:t_i \in S_j} u_i \leq \epsilon c'(S_j))$. (This condition will let us prove that $c'(S_j) \leq (\sum_{i:t_i \in S_j} u_i)/(1-\epsilon)$.) We stop as soon as we have covered at least k elements. Suppose the algorithm covers at least k elements after ℓ rounds. The main problem is that in the last round we can include many sets simultaneously, while we can afford to include only a few. Let δ be the number of elements that we need to cover *after* round $\ell - 1$. To select an appropriate subset of the chosen sets, we need to pick a minimal collection of chosen sets that cover at least δ elements. To accomplish this, we order the sets chosen in the last iteration arbitrarily. Now compute in parallel the "effective" number of elements each set covers and choose a minimal collection based on the fixed ordering. (All these steps can be implemented in parallel using prefix computations.)

Theorem 2. *The parallel algorithm runs in* $(1 + f \log(1/\epsilon))(1 + \log n)$ *rounds, with each round running in* $O(\log n)$ *time; the number of processors is linear in the size of the input. The algorithm produces an* $\frac{f}{1-\epsilon}$*-approximate solution.*

3 Set Cover for Small Sets

Problem: Given a collection C of small subsets of a base set U. Each small subset in the collection has size at most Δ, and their union is U. The objective is to find a minimum size sub-collection that covers at least k elements.

Here we have the original partial set cover instance with the additional information that the sets are of "small" size, i.e., Δ is small. We obtain an approximation factor of $4/3$ for the case when $\Delta = 3$ using the the idea of (s,t) semi-local optimization [11]. This technique consists of inserting up to s 3-sets (sets of size 3) and deleting up to t 3-sets from the current cover. Then the elements that are not covered by the 3-sets (already existing ones + the newly added) are covered optimally using 2-sets and 1-sets. This can be solved in polynomial time using maximum matching [13]. The vertices are the uncovered elements of U and the edges are the admissible 2-sets. The 2-sets corresponding to the maximum matching edges and the 1-sets corresponding to the vertices not covered by the maximum matching form an optimum covering. We will order the quality of a solution by the number of sets in the cover and among two covers of the same size we choose the one with fewer 1-sets and if the covers have the same size and neither cover has a 1-set we choose the one that covers more elements.

The algorithm starts with any solution. One solution can be obtained as follows. Choose a maximal collection of disjoint 3-sets. Cover the remaining elements optimally using 2-sets and 1-sets. Perform semi-local $(2,1)$ improvements until no improvement is possible.

The proof for the bound of $4/3$ for full coverage does not extend to the partial coverage version. For the full coverage, to prove the lower bound on the optimal

solution Duh and Fürer construct a graph G in which the vertices are the sets chosen by OPT and the edges are 1-sets and 2-sets of the approximate solution. They prove that G can not have more than one cycle and hence argue that the total number of 1-sets and 2-sets in the solution is a lower bound on OPT. This works well for the full coverage version but breaks down for the partial covering problem. For the partial covering case G having at most one cycle is a necessary but not a sufficient condition to prove the lower bound.

In the full version of the problem, to bound the number of 1-sets in the solution they construct a bipartite graph with the two sets of vertices corresponding to the sets chosen by the approximate solution and OPT. If a set corresponding the approximate solution intersects a set corresponding to OPT in m elements then there are m edges between their corresponding vertices in the graph. In each component of the graph they show that the number of 1-sets of the solution in that component is at most the number of 1-sets of OPT in that component. This is clearly not the case in the partial covering case. We obtain a bound on the number of 1-sets as a side effect of the proof for the lower bound on OPT.

Theorem 3. *The semi-local* $(2,1)$*-optimization algorithm for* 3*-set partial covering problem produces a solution that is within* $\frac{4}{3}OPT + 1$.

4 Probabilistic Approaches for *k*-Vertex Cover

We now present a randomized rounding approach to the natural LP relaxation of k-vertex cover. Analyzed in three different ways, this leads to three new approximation results mentioned in §1: relating to vertex cover (full coverage) for expander graphs of constant average degree, k-vertex cover on bounded-degree graphs, and multi-criteria k-vertex cover problems. The k-vertex cover problem on a graph $G = (V, E)$ can be formulated as an integer program as follows. We assign binary variables x_j for each $v_j \in V$ and $z_{i,j}$ for each $(i,j) \in E$. Here, $x_j = 1$ iff vertex v_j belongs to the cover, and $z_{i,j} = 1$ iff edge (i,j) is covered. The LP relaxation is obtained by letting each x_j and $z_{i,j}$ lie in $[0,1]$:

$$\min \quad \sum_{j=1}^{n} x_j \quad \text{subject to}$$

$$x_i + x_j \geq z_{i,j}, \quad (i,j) \in E \tag{1}$$

$$\sum_{(i,j)\in E} z_{i,j} \geq k \tag{2}$$

$$x_j, z_{i,j} \in [0,1], \quad \forall i,j.$$

Our basic approximation recipe will be as follows. The LP relaxation is solved optimally. Let $\{x_i^*\}$, $\{z_{i,j}^*\}$ denote an optimal LP solution, and let $\lambda = 2(1-\epsilon)$, where $\epsilon \in [0, 1/2]$ is a parameter that will be chosen based on the application. Let $S_1 = \{v_j | x_j^* \geq 1/\lambda\}$, and $S_2 = V - S_1$. Include all the vertices in S_1 as

part of our cover, and mark the edges incident on vertices in S_1 as covered. Now independently for each $j \in S_2$, round x_j to 1 with a probability of λx_j^*, and to 0 with a probability of $1 - \lambda x_j^*$. Let W be the random variable denoting the number of covered edges at this point. If $W < k$, we choose any $k - W$ uncovered edges and cover them by arbitrarily choosing one end-point for each of them.

We now introduce some notation to analyze the above process. Throughout, we let $\Pr[\cdot]$ and $\mathbf{E}[\cdot]$ denote probability and expectation, respectively. Let y^* represent the optimal objective function value of the LP, and define $S_0 \subseteq S_1$ by $S_0 = \{v_j : x_j^* = 1\}$. Let y_F^* and y_P^* be the contribution to y^* of the vertices in S_0 and $V - S_0$ respectively. Denote by $U_{i,j}$ the event that edge (i, j) is uncovered. Let C_1 be the cost of the solution produced by our randomized scheme *before* the step of covering $k - W$ edges if necessary, and let C_2 be the cost incurred in covering these $k - W$ edges, if any. The total cost C is of course $C_1 + C_2$; thus, $\mathbf{E}[C] = \mathbf{E}[C_1] + \mathbf{E}[C_2]$. Now, it is easy to check that $\mathbf{E}[C_1] \leq y_F^* + \lambda y_P^*$, and that $\mathbf{E}[C_2] \leq \mathbf{E}[\max\{k - W, 0\}]$. So we have

$$\mathbf{E}[C] \leq y_F^* + \lambda y_P^* + \mathbf{E}[\max\{k - W, 0\}]. \tag{3}$$

As usual, let $\overline{\mathcal{E}}$ denote the complement of an event $\mathcal{E}$. Lemma 1 on the statistics of W will be useful; we only give a proof sketch here.

Lemma 1. *(i)* $\mathbf{E}[W] \geq k(1-\epsilon^2)$. *(ii) Suppose the graph G has maximum degree d. Then, the variance $Var[W]$ of W is at most $d\mathbf{E}[W]$.*

Proof. (i) Consider any edge (i, j). Now if $x_i^* \geq 1/\lambda$ or $x_j^* \geq 1/\lambda$, $\Pr[U_{i,j}] = 0$; otherwise, $\Pr[U_{i,j}] = (1 - \lambda x_i^*)(1 - \lambda x_j^*)$. In the latter case, since $x_i^* + x_j^* \geq z_{i,j}^*$ and $z_{i,j}^* \in [0, 1]$, we can show that $\Pr[U_{i,j}] \leq (1 - \lambda z_{i,j}^*/2)^2 \leq 1 - z_{i,j}^*(1 - \epsilon^2)$. Since $\mathbf{E}[W] = \sum_{(i,j)\in E} \Pr[\overline{U_{i,j}}]$, we get $\mathbf{E}[W] \geq k(1 - \epsilon^2)$.
(ii) We have $W = \sum_{(i,j)\in E} \overline{U_{i,j}}$. It can be checked that if a random variable W' is the sum of *pairwise independent* random variables each of which lies in $[0, 1]$, then $\mathrm{Var}[W'] \leq \mathbf{E}[W']$. However, the terms $U_{i,j}$ that constitute W do have some dependent pairs: if edges (i, j) and (i', j') share an endpoint, then $\overline{U_{i,j}}$ and $\overline{U_{i',j'}}$ are dependent. Define γ to be the sum, over all unordered pairs of distinct edges (i, j) and (i', j') that share an end-point, of $\Pr[\overline{U_{i,j}} \wedge \overline{U_{i',j'}}]$. Using the above observations and the definition of variance, we can show that $\mathrm{Var}[W] \leq \mathbf{E}[W] + \gamma$. Now, for any term $p \doteq \Pr[\overline{U_{i,j}} \wedge \overline{U_{i',j'}}]$ in γ, $p \leq \min\{\Pr[\overline{U_{i,j}}], \Pr[\overline{U_{i',j'}}]\} \leq (\Pr[\overline{U_{i,j}}] + \Pr[\overline{U_{i',j'}}])/2$. Finally, since each edge has at most $2(d-1)$ other edges that share an end-point with it, we get

$$\mathrm{Var}[W] \leq \mathbf{E}[W] + \gamma \leq \mathbf{E}[W] + \sum_{(i,j)\in E} (2(d-1)/2) \cdot \Pr[\overline{U_{i,j}}] = d\mathbf{E}[W].$$

4.1 Vertex Cover on Expanders

Suppose we have a vertex cover problem; i.e., k-vertex cover with $k = m$. The LP relaxation here has "1" in place of "$z_{i,j}$" in (1), and does not require the

variables $z_{i,j}$ and the constraint (2). We focus here on the case of expander graphs of constant average degree. That is, for some constants c and d, we are studying graphs where: (i) the number of edges m is at most nd, and (ii) for any set X of vertices with $|X| \leq n/2$, at least $c|X|$ vertices outside X have a neighbor in X. Since $k = m$, it is well-known that we can efficiently compute an optimal solution x^* to the LP with all entries lying in $\{0, 1/2, 1\}$. Let $H = \{v_j | x^*_j = 1/2\}$ and $F = \{v_j | x^*_j = 1\}$. Also, since $W \leq k = m$ always holds, $\mathbf{E}[\max\{k - W, 0\}] = \mathbf{E}[k - W] \leq m\epsilon^2$, by Lemma 1(i). Thus, (3) shows that $\mathbf{E}[C]$ is at most $y^*_F + 2(1-\epsilon)y^*_H + m\epsilon^2$. (The overall approach of: (i) conducting a randomized rounding and then doing a greedy fixing of violated constraints, and (ii) using an equality such as our "$\mathbf{E}[\max\{k - W, 0\}] = \mathbf{E}[k - W]$" here, is suggested in [30]. We next show how expansion is useful in bounding $\mathbf{E}[C]$ well. However, in the context of *partial* covering, an equality such as "$\mathbf{E}[\max\{k - W, 0\}] = \mathbf{E}[k - W]$" does not hold; so, as discussed in §4.2 and §4.3, new analysis approaches are employed there.) Choosing $\epsilon = y^*_H/m$, we get

$$\mathbf{E}[C] \leq y^*_H(2 - y^*_H/m) + y^*_F. \tag{4}$$

Case I: $|H| \leq n/2$**.** By the LP constraints, the edges incident on vertices in H must have their other end-point in F. Since G is an expander, $|F| \geq c \cdot |H|$. Also, $y^*_F = |F|$ and $y^*_H = |H|/2$. So, since $y^* = y^*_H + y^*_F$, we have $y^*_H = y^*/(1+a)$ for some $a \geq 2c$. We can now use (4) to get

$$\mathbf{E}[C] \leq 2y^*_H + y^*_F = (2 - a/(1+a))y^* \leq (2 - 2c/(1+2c))y^*.$$

Case II: $|H| > n/2$**.** So, we have $y^*_H \geq n/4$. Bound (4) shows that $\mathbf{E}[C] \leq (2 - y^*_H/m)y^*$; we have $m \leq nd$ by assumption. So, $\mathbf{E}[C] \leq (2 - 1/(4d))y^*$.

Thus we see that $\mathbf{E}[C] \leq [2 - \min\{2c/(1+2c), 1/(4d)\}] \cdot y^*$; i.e., we get a constant-factor approximation that is strictly better than 2.

4.2 k-Vertex Cover: Bounded-Degree Graphs

We now show that any constant d, k-vertex cover on graphs of maximum degree at most d can be approximated to within $2(1 - \Omega(1/d))$, for any value of the parameter k. We also prove that the integrality gap in this case is at most $2(1-\Omega(1/d))$. We start with a couple of useful tail bounds. First, suppose X is a sum of independent random variables X_i each of which lies in $[0, 1]$; let $\mathbf{E}[X] = \mu$. Then for any $\delta \in [0, 1]$, the Chernoff bound shows that $\Pr[X \geq \mu(1+\delta)]$ is at most $e^{-\mu\delta^2/3}$. Next, suppose X is a random variable with mean μ and variance σ^2; suppose $a > 0$. The Chebyshev-Cantelli inequality (see, e.g., [1]), shows that $\Pr[X - \mu \leq -a] \leq \sigma^2/(\sigma^2 + a^2)$. We now analyze the performance of our basic algorithm (of randomized rounding of the LP solution followed by a simple covering of a sufficient number of edges), for the k-vertex cover problem on graphs with maximum degree bounded by some given constant d. The notation remains the same. The main problem in adopting the method of §4.1 here is as follows. Since k equaled m there, we could use the equality $\mathbf{E}[\max\{k-W, 0\}] = \mathbf{E}[k-W]$,

thus substantially simplifying the analysis. Such an equality is not true here; also, $\mathbf{E}[\max\{X, 0\}] \geq \max\{\mathbf{E}[X], 0\}$ for any random variable X. (The two sides of this inequality may differ a lot: if X is the sum of n i.i.d. random variables each of which is uniformly distributed on $\{-1, 1\}$, then the r.h.s. is zero, while the l.h.s. is $\Theta(\sqrt{n})$.) However, Lemma 1, Chebyshev-Cantelli, and a case analysis of whether $k \geq 4d$ or not, can be used to show

$$\Pr[W \leq (k(1-\epsilon^2) - 2\sqrt{kd})] \leq 1/5. \tag{5}$$

Next, for a suitably large constant c_0, we can assume that $k \geq c_0 d^5$. (Any optimal solution has size at most k, since in an optimal solution, every vertex should cover at least one new edge. So if k is bounded by a constant–such as $c_0 d^5$–then we can find an optimal solution in polynomial time by exhaustive search.) Also, by adding all the constraints of the LP and simplifying, we get that $y^* \geq k/d$. Thus, letting $\delta = 1/(3d)$, a Chernoff bound shows that immediately after the randomized rounding, the probability of having more than $2y^*(1-\epsilon)(1+\delta)$ vertices in our initial cover is at most $1/5$ (if the constant c_0 is chosen large enough). Recall (5). So, with probability at least $1-(1/5+1/5) = 3/5$, the final cover we produce is of size at most $2y^*(1-\epsilon)(1+\delta) + k\epsilon^2 + 2\sqrt{kd}$. We now choose $\epsilon = y^*(1+\delta)/k$; since $y^* \geq k/d \geq c_0 d^4$ with c_0 sufficiently large, some simplification shows that the final cover size is at most $2y^*(1 - \Omega(1/d))$.

4.3 k-Vertex Cover: Multiple Criteria

We now briefly consider multi-criteria k-vertex cover problems on arbitrary graphs. Here, we are given a graph G and, as usual, have to cover at least k edges. We are also given ℓ "weight functions" w_i, and want a cover that is "good" w.r.t. all of these. More precisely, suppose we are given vectors $w_i \in [0,1]^n$, $i = 1, 2, \ldots, \ell$, and a fractional solution x^* to the k-cover problem on G. Let $w_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,n})$, and define $y_i^* = \sum_j w_{i,j} x_j^*$ for $1 \leq i \leq \ell$. We aim for an integral solution z such that for *each* i, $y_i = \sum_j w_{i,j} z_j$ is not "much above" y_i^*. Multi-criteria optimization has recently received much attention, since participating individuals/organizations may have differing objective functions, and we may wish to (reasonably) simultaneously satisfy all of them if possible. The result we show here is that if $y_i^* \geq c_1 \log^2(\ell + n)$ for all i (where c is a sufficiently large constant), then we can efficiently find an integral solution z with $y_i \leq 2(1 + 1/\sqrt{\log(\ell+n)})y_i^*$ for each i.

5 Geometric Packing and Covering

Recall this problem's definition from §1. A polynomial-time approximation scheme exists for the case when $k = n$ (full covering). The algorithm uses a strategy, called the *shifting strategy*. The strategy is based on a divide and conquer approach. The area, I, enclosing the set of given points is divided into strips of width D. Let l be the shifting parameter. Groups of l consecutive strips, resulting in strips of width lD are considered. For any fixed subdivision of I into

strips of width D, there are l different ways of partitioning I into strips of width lD. The l partitions are denoted by $S_1, S_2, \ldots, S_l$. The solution to cover all the points is obtained by finding the solution to cover the points for each partition, $S_j, 1 \leq j \leq l$, and then choosing a minimum cost solution. A solution for each partition is obtained by finding a solution to cover the points in each strip (of width lD) of that partition and then taking the union of all such solutions. To obtain a solution for each strip, the shifting strategy is re-applied to each strip. This results in the partition of each strip into "squares" of side length lD. As will be shown later, there exists an optimal covering for such squares.

We modify the use of shifting strategy for the case when $k \leq n$ (partial covering). The obstacle in directly using the shifting strategy for the partial covering case is that we do not know the number of points that an optimal solution covers in each strip of a partition. This is not a problem with the full covering case because we know that any optimal solution would have to cover all the points within each strip of a partition. For the partial covering, this problem is overcome by "guessing" the number of points covered by an optimal solution in each strip. This is done by finding a solution for every possible value for the number of points that can be covered in each strip and storing each solution. A formal presentation is given below.

Let A be any algorithm that delivers a solution to cover the points in any strip of width lD. Let $A(S_i)$ be the algorithm that applies A to each strip of the partition S_i and outputs the union of all disks in a feasible solution. We will find such a solution for each of the l partitions and output the minimum. Consider a partition S_i containing p strips of width lD. Let n_j be the number of points in strip j. Let n_j^{OPT} be the number of points covered by OPT in strip j. Since we do not know n_j^{OPT}, we will find feasible solutions to cover points for all possible values of n_j^{OPT}. Note that $0 \leq n_j^{OPT} \leq k'_j = \min(k, n_j)$. A dynamic programming formulation is as follows:

$$C(x, y) = \min_{0 \leq i \leq k'_x} (D_i^x + C(x-1, y-i))$$

where $C(x, y)$ denotes the number of disks needed to cover y points in strips $1..x$ and D_i^x is the number of disks needed to cover i points in strip x. Computing $C(p, k)$ gives us the desired answer.

For each strip s, for $0 \leq i \leq k'_s, D_i^s$ can be calculated by recursive application of the algorithm to the strip s. We partition the strip into squares of side length lD. We can find optimal coverings of points in such a square by exhaustive search. With $O(l^2)$ disks of diameter D we can cover $lD \times lD$ square compactly, thus we never need to consider more disks for one square. Further, we can assume that any disk that covers at least two of the given points has two of these points on its border. Since there are only two ways to draw a circle of given diameter through two given points, we only have to consider $2\binom{n'}{2}$ possible disk positions where n' is the number of given points in the considered square. Thus, we have to check for at most $O(n'^{2(l\sqrt{2})^2})$ arrangements of disks.

Let Z^A be the value of the solution delivered by algorithm A. The shift algorithm S_A is defined for a local algorithm A. Let r_B denote the performance ratio of an algorithm B; that is, r_B is defined as the supremum of $Z^B/|OPT|$ over all problem instances. We can show:

Lemma 2. *$r_{S_A} \leq r_A(1+\frac{1}{l})$ where A is the local algorithm and l is the shifting parameter.*

Theorem 4. *The above algorithm yields a PTAS with performance ratio at most $(1+\frac{1}{l})^2$.*

Proof. We use two nested applications of the shifting strategy to solve the problem. The above lemma applied to the first application of the shifting strategy would relate the performance ratio of the final solution, r_{S_A}, to that of the solution for each strip, r_A: $r_{S_A} \leq r_A(1+1/l)$. The lemma when applied to the second application of shifting strategy relates r_A to the performance ratio of the solution to each square, say $r_{A'}$. Thus, $r_A \leq r_{A'}(1+1/l)$. But since we obtain an optimal solution for each square, $r_{A'} = 1$. Thus we have $r_{S_A} \leq (1+1/l)^2$.

Acknowledgements. We thank the referees for their helpful feedback. Part of this work was done when the second and third authors attended the DIMACS Workshop on *Multimedia Streaming on the Internet* at the DIMACS Center, Rutgers University, Piscataway, NJ, on June 12–13, 2000.

References

1. N. Alon, R. Boppana and J. H. Spencer. An asymptotic isoperimetric inequality. *Geometric and Functional Analysis*, 8:411–436, 1998.
2. B. Baker. Approximation Algorithms for NP-Complete Problems on Planar Graphs. *JACM*, Vol 41 (1), (1994), pp. 153–190.
3. R. Bar-Yehuda and S. Even. A linear time approximation algorithm for the weighted vertex cover problem. *J. of Algorithms* 2:198-203, 1981.
4. R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27-45, 1985.
5. R. Bar-Yehuda. Using homogeneous weights for approximating the partial cover problem. In *Proc. Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 71-75, 1999.
6. N. Bshouty, and L. Burroughs. Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. *The Proceedings of the Fifteenth Annual Symposium on the Theoretical Aspects of Computer Science* 298-308, 1998.
7. M. Charikar, S. Khuller, D. Mount, and G. Narasimhan. Algorithms for Facility Location Problems with Outliers. In *Proc. Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, 642-651, 2001.
8. V.Chvátal. A greedy heuristic for the set-covering problem. *Math. of Oper. Res.* Vol. 4, 3, 233-235, 1979.
9. K. L. Clarkson. A modification of the greedy algorithm for the vertex cover. *Information Processing Letters* 16:23-25, 1983.

10. T. H. Cormen, C. E. Leiserson and R. L. Rivest, "Introduction to Algorithms", *MIT Press*, 1989.
11. R. Duh and M. Fürer. Approximating k-set cover by semi-local optimization. In *Proc. 29th STOC*, May 1997, pages 256–264.
12. R. Gandhi, S. Khuller and A. Srinivasan. Approximation algorithms for partial covering problems. *Technical Report CS-TR-# 4234* (April 2001). Also available at: http://www.cs.umd.edu/users/samir/grant/icalp01.ps
13. O. Goldschmidt, D. Hochbaum, and G. Yu. A modified greedy heuristic for the set covering problem with improved worst case bound. *Information Processing Letters* 48(1993), 305-310.
14. M. X. Goemans and J. Kleinberg. The Lovász theta function and a semidefinite programming relaxation of vertex cover. *SIAM Journal on Discrete Mathematics*, 11:196–204, 1998.
15. M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296-317, 1995.
16. M. Halldórsson. Approximating k-set cover and complementary graph coloring. In *Proc. Fifth Conference on Integer Programming and Combinatorial Optimization*, June 1996, LNCS 1084, pages 118–131.
17. E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. In *Proc. Eleventh ACM-SIAM Symposium on Discrete Algorithms*, January 2000, pages 329–337.
18. D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. W.P.#64-79-80, GSIA, Carnegie-Mellon University, April 1980. Also: *SIAM J. Comput.* 11(3) 1982.
19. D. S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics* 6:243-254, 1983.
20. D. S. Hochbaum (editor). Approximation Algorithms for NP-hard problems. *PWS Publishing Company,* 1996.
21. D. S. Hochbaum. The t-vertex cover problem: Extending the half integrality framework with budget constraints. In *Proc. First International Workshop on Approximation Algorithms for Combinatorial Optimization Problems* 111-122, 1998.
22. D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of ACM*, 32(1):130-136, 1985.
23. D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9:256-278, 1974.
24. M. Kearns. The computational complexity of machine learning. *M.I.T. Press*, 1990.
25. S. Khuller, U. Vishkin, and N. Young. A Primal Dual Parallel Approximation Technique Applied to Weighted Set and Vertex Cover. *Journal of Algorithms*, 17(2):280–289, 1994.
26. L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Math.* 13:383-390, 1975.
27. G. L. Nemhauser and L. E. Trotter, Jr. Vertex packings: Structural properties and algorithms. *Mathematical Programming* 8:232-248, 1975.
28. E. Petrank. The hardness of approximation: Gap location. *Computational Complexity* 4:133-157, 1994.
29. P. Slavík. Improved performance of the greedy algorithm for partial cover. *Information Processing Letters* 64:251-254, 1997.
30. A. Srinivasan. New Approaches to Covering and Packing Problems. In *Proc. Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, 567-576, 2001.

On the Online Bin Packing Problem

Steven S. Seiden*

Department of Computer Science
298 Coates Hall
Louisiana State University
Baton Rouge, LA 70803, USA
sseiden@acm.org

Abstract. A new framework for analyzing online bin packing algorithms is presented. This framework presents a unified way of explaining the performance of algorithms based on the HARMONIC approach [3,5,8,10,11,12]. Within this framework, it is shown that a new algorithm, HARMONIC++, has asymptotic performance ratio at most 1.58889. It is also shown that the analysis of HARMONIC+1 presented in [11] is incorrect; this is a fundamental logical flaw, not an error in calculation or an omitted case. The asymptotic performance ratio of HARMONIC+1 is at least 1.59217. Thus HARMONIC++ provides the best upper bound for the online bin packing problem to date.

Keywords: bin packing, online algorithms.

1 Introduction

Bin packing is one of the oldest and most well studied problems in computer science [4,12]. The study of this problem dates back to the early 1970's, when computer science was still in its formative phase—ideas which originated in the study of the bin packing problem have helped shape computer science as we know it today. The influence and importance of this problem are witnessed by the fact that it has spawned off whole areas of research, including the fields of online algorithms and approximation algorithms.

Problem Definition: In the *bin packing* problem, we receive a sequence σ of *pieces* $p_1, p_2, \ldots, p_N$. We use the words piece and *item* synonymously. Each piece has a fixed *size* in $(0, 1]$. In a slight abuse of notation, we use p_i to indicate both the ith piece and its size. The usage should be obvious from the context. We have an infinite number of *bins* each with *capacity* 1. Each piece must be assigned to a bin. Further, the sum of the sizes of the items assigned to any bin may not exceed its capacity. A bin is *empty* if no piece is assigned to it, otherwise it is *used*. The goal is to minimize the number of bins used.

* This research was partially supported by an LSU Council on Research summer stipend and by the Research Competitiveness Subprogram of the Louisiana Board of Regents.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 237–248, 2001.

In the *online* version of this problem, each piece must be assigned in turn, without knowledge of the next pieces. Since it is impossible in general to produce the best possible solution when computation occurs online, we consider approximation algorithms. Basically, we want to find an algorithm which uses a number of bins which is within a constant factor of the minimum possible number, no matter what the input is. This constant factor is known as the asymptotic performance ratio.

We define the asymptotic performance ratio more precisely. For a given input sequence σ, let $\mathrm{cost}_{\mathcal{A}}(\sigma)$ be the number of bins used by algorithm $\mathcal{A}$ on σ. Let $\mathrm{cost}(\sigma)$ be the minimum possible number of bins used to pack pieces in σ. The *asymptotic performance ratio* for an algorithm $\mathcal{A}$ is defined to be

$$R_{\mathcal{A}}^{\infty} = \limsup_{n\to\infty} \sup_{\sigma} \left\{ \frac{\mathrm{cost}_{\mathcal{A}}(\sigma)}{\mathrm{cost}(\sigma)} \middle| \mathrm{cost}(\sigma) = n \right\}.$$

Let $\mathcal{O}$ be the set of all online bin packing algorithms. The *optimal asymptotic performance* ratio is defined to be $R_{\mathrm{OPT}}^{\infty} = \inf_{\mathcal{A}\in\mathcal{O}} R_{\mathcal{A}}^{\infty}$. Our goal is to find an algorithm with asymptotic performance ratio close to $R_{\mathrm{OPT}}^{\infty}$.

Previous Results: The online bin packing problem was first investigated by Johnson [6]. He showed that the NEXT FIT algorithm has performance ratio 2. Subsequently, it was shown by Johnson, Demers, Ullman, Garey and Graham that the FIRST FIT algorithm has performance ratio $\frac{17}{10}$ [7]. Yao showed that REVISED FIRST FIT has performance ratio $\frac{5}{3}$, and further showed that no online algorithm has performance ratio less than $\frac{3}{2}$ [14]. Brown and Liang independently improved this lower bound to 1.53635 [1,9]. The lower bound currently stands at 1.54014, due to van Vliet [13]. Define $u_{i+1} = u_i(u_i - 1) + 1$, $u_1 = 2$ and

$$h_{\infty} = \sum_{i=1}^{\infty} \frac{1}{u_i - 1} \approx 1.69103.$$

Lee and Lee showed that the HARMONIC algorithm, which uses bounded space, achieves a performance ratio arbitrarily close to h_{∞} [8]. They further showed that no bounded space online algorithm achieves a performance ratio less than h_{∞} [8]. In addition, they developed the REFINED HARMONIC algorithm, which they showed to have a performance ratio of $\frac{373}{228} < 1.63597$. The next improvements were MODIFIED HARMONIC and MODIFIED HARMONIC 2. Ramanan, Brown, Lee and Lee showed that these algorithms have performance ratios of $\frac{538}{333} < 1.61562$ and $\frac{239091}{148304} < 1.61217$, respectively [10]. The best result to date is that of Richey [11]. He presents an algorithm called HARMONIC+1 and claims that it has performance ratio 1.58872.

Our Results: In this paper, we present a general framework for analyzing a large class of online bin packing algorithms. This class includes HARMONIC, REFINED HARMONIC, MODIFIED HARMONIC, MODIFIED HARMONIC 2 and HARMONIC+1. In fact, we show that all these algorithm are just special cases of a general algorithm which we call SUPER HARMONIC. We present a general

analysis of SUPER HARMONIC. Our analysis is qualitatively different than previous ones, in that we reduce the problem of analyzing an instance of SUPER HARMONIC to that of solving a specific knapsack problem instance. We develop a branch an bound algorithm for solving such knapsack problems. Thus we provide a general computer assisted method of proving upper bounds for all algorithms that can be expressed in terms of SUPER HARMONIC. This leads us to fundamental logical flaw in the analysis of HARMONIC+1. We show that the performance ratio of HARMONIC+1 is at least 1.59217. In light of this finding, we develop a new algorithm called HARMONIC++, and show that it has asymptotic performance ratio at most 1.58889. Thus HARMONIC++ provides the best upper bound for the online bin packing problem to date. We also note that 1.58333 is a lower bound for any SUPER HARMONIC algorithm, thus HARMONIC++ has performance reasonably close to the best possible SUPER HARMONIC algorithm.

Due to space constraints, several proofs and a full description of the algorithm are omitted. They can be found in an appendix, available at:

`http://www.csc.lsu.edu/~seiden/append.ps.Z`

2 Interval Classification Algorithms

An interval classification algorithm operates by classifying pieces according to a set of predefined intervals. Let $t_1 = 1 > t_2 > \cdots > t_n > t_{n+1} > 0$ be real numbers. We define $\epsilon = t_{n+1}$ and $t_{n+2} = 0$. The interval I_j is defined to be $(t_{j+1}, t_j]$ for $j = 1, \ldots, n+1$. Note that these intervals are disjoint and that they cover $(0, 1]$. A piece of size s has *type* j if $s \in I_j$.

The NEXT FIT algorithm [6] is used to pack all items of size at most ϵ. The algorithm maintains a single *open* bin. If the current item fits into the open bin, it is placed there. Otherwise, the open bin is *closed* and a new open bin is allocated. Obviously, this algorithm is online, runs in linear time and uses constant space. The following well known lemma shall prove useful:

Lemma 1. *If the sum of the sizes of the items packed by* NEXT FIT *is x, and each item has size at most ϵ, then the number of bins used is at most $x/(1-\epsilon)+1$.*

Proof. Every bin packed by NEXT FIT, except possibly the one open bin, contains pieces whose total size is at least $1-\epsilon$. Therefore, the total number of bins used is at most $\lceil x/(1-\epsilon) \rceil \leq x/(1-\epsilon) + 1$. □

A *packing* is a tuple $q = \langle q_1, \ldots, q_n \rangle$ over $\mathbb{N}$ such that $\sum_{i=1}^{n} q_i t_i \leq 1$. Intuitively, a packing describes the contents of one of the algorithm's bins. I.e. q_i is the number of items of type i contained in the bin. All interval classification algorithms operate by placing items according to some predetermined set of packings. We call the set of bins with a particular packing a *group*.

An important subset of interval classification algorithms can be described in terms of one general algorithm, which we call SUPER HARMONIC. All of the algorithms considered here fall into this sub-class.

An instance of the SUPER HARMONIC algorithm is described by the following parameters: integers n and K; real numbers $1 > t_2 > \cdots > t_n > t_{n+1} = \epsilon > 0$, $\alpha_1, \ldots, \alpha_n, \in [0,1]$ and $0 < \Delta_1 < \cdots < \Delta_K < \frac{1}{2}$; and a function $\phi : \{1, \ldots, n\} \mapsto \{0, \ldots, K\}$. Define $t_1 = 1$ and $\Delta_0 = 0$. In the following paragraphs, we describe the operation of SUPER HARMONIC.

Upon receipt, each item of type $i \leq n$ is assigned a color, *red* or *blue*. The algorithm uses two sets of counters, $e_1, \ldots, e_n$ and $s_i, \ldots, s_n$, all of which are initially zero. The total number of type i items is s_i, while the number of type i red items is e_i. For $1 \leq i \leq n$, the invariant $e_i = \lfloor \alpha_i s_i \rfloor$ is maintained.

$\beta_i = \lfloor 1/t_i \rfloor$ is the number of type i items which fit in a bin. Blue items of type i are placed β_i in a bin, as in HARMONIC.

$\delta_i = 1 - t_i\beta_i$ is the amount of space left when a bin is filled with β_i type i items. If possible, we would like to use this space to pack red items. We require that ϕ satisfy $\Delta_{\phi(i)} \leq \delta_i$. Intuitively, $\mathcal{D} = \{\Delta_1, \ldots, \Delta_K\}$ describes the set of spaces into which red items can be placed. $\Delta_{\phi(i)}$ is the amount of space used to hold red items in a bin which holds blue items of type i. $\phi(i) = 0$ indicates that no red items can be accepted. To ensure that every red item potentially can be placed, we require that $\alpha_i = 0$ for all i such that $t_i > \Delta_K$. Define $\gamma_i = 0$ if $t_i > \Delta_K$ and $\gamma_i = \max\{1, \lfloor \Delta_1/t_i \rfloor\}$ otherwise. This is the number of red items of type i that the algorithm places together in a bin. Note that this is the maximum number guaranteed to fit in every space in $\mathcal{D}$. Define

$$\varphi(i) = \min\{j \mid t_i \leq \Delta_j, 1 \leq j \leq K\}.$$

Intuitively, $\varphi(i)$ is the index of the smallest space in $\mathcal{D}$ into which a red item of type i can be placed.

The bin groups used are named:

$$\{i \mid \phi_i = 0,\ 1 \leq i \leq n,\},$$
$$\{(i,?) \mid \phi_i \neq 0,\ 1 \leq i \leq n,\},$$
$$\{(?,j) \mid \alpha_i \neq 0,\ 1 \leq i \leq n,\},$$
$$\{(i,j) \mid \phi_i \neq 0,\ \alpha_j \neq 0,\ \gamma_j t_j \leq \Delta_{\phi(i)},\ 1 \leq i \leq n,\ 1 \leq j \leq n\}.$$

We call these groups *monochromatic*, *indeterminate blue*, *indeterminate red* and *bichromatic*, respectively. Collectively, we call the monochromatic and bichromatic groups *final* groups.

The monochromatic group i contains bins which hold only blue items of type i. There is one open bin in each of these groups; this bin has fewer than β_i items. The closed bins all contain β_i items.

The bichromatic group (i,j) contains bins which contain blue items of type i along with red items of type j. A closed bin in this group contains β_i type i items and γ_j type j items. There are at most three open bins.

The indeterminate blue group $(i,?)$ contains bins which hold only blue items of type i. These bins are all open, but only one has fewer than β_i items.

The indeterminate red group $(?,j)$ contains bins which hold only red items of type j. Again, these bins are all open, but only one has fewer than γ_j items.

Essentially, the algorithm tries to minimize the number of indeterminate bins, while maintaining all the aforementioned invariants. I.e. we try to place red and blue items together whenever possible; when this is not possible we place them in indeterminate bins in hope that they can later be so combined. A formal description of Super Harmonic is displayed in Figure 1. The symbols ♢, ♣, ⋆, ♡, ♣ and † are labels used in the proof of Lemma 2.

Initialize $e_i \leftarrow 0$ and $s_i \leftarrow 0$ for $1 \leq i \leq n$.
For each piece p:
  $i \leftarrow$ type of p.
  If $i = n + 1$ place p using Next Fit.
  Else:
    $s_i \leftarrow s_i + 1$.
    If $e_i < \lfloor \alpha_i s_i \rfloor$:
      $e_i \leftarrow e_i + 1$.
      Color p red.
      ♢ If, for any j, there is an open bin in group (j, i) or $(?, i)$ with fewer than γ_i type i items, then place p in this bin.
      ♣ Else if there is some bin in group $(j, ?)$ such that $\Delta_{\phi(j)} \geq \gamma_i t_i$, then place p in it and change the group of this bin to (j, i).
      ⋆ Otherwise, open a new group $(?, i)$ bin and place p in it.
    Else:
      Color p blue.
      If $\phi_i = 0$:
        If there is an open bin in group i with fewer than β_i items, then place p in this bin.
        If not, open a new group i bin and place p there.
      Else:
        ♡ If, for any j, there is an open bin in group (i, j) with fewer than β_i type i items, then place p in this bin.
        Else if there is an open bin in group $(i, ?)$ with fewer than β_i type i items, then place p in this bin.
        ♠ Else if there is some bin in group $(?, j)$ such that $\Delta_{\phi(i)} \geq \gamma_j t_j$ then place p in it and change the group of this bin to (i, j).
        † Otherwise, open a new group $(i, ?)$ bin and place p there.

Fig. 1. The Super Harmonic Algorithm.

Lemma 2. Super Harmonic *maintains the following invariants: (1) The number of red items of type i is $\lfloor \alpha_i s_i \rfloor$. (2) At most one bin is open in any group i. (3) At most three bins are open in any group (i, j). (4) At most one bin has fewer than β_i items in any group $(i, ?)$. (5) At most one bin has fewer than γ_i items in any group $(?, i)$.*

Due to space considerations, the proof is given in the appendix.

Corollary 3.1 of Ramanan *et al.* [10] implies the following result:

Lemma 3 (Ramanan *et al.*). *For all choices of parameters, the asymptotic performance ratio of* Super Harmonic *is at least* $\frac{19}{12} > 1.58333$.

3 Weighting Systems

Analysis based on *weighting functions* is introduced in [7], and is used in the subsequent work on interval classification bin packing algorithms [8,10,11]. We generalize the idea of a weighting function here.

Let $\mathbb{R}$ and $\mathbb{N}$ be the sets of real numbers and non-negative integers, respectively.

A *weighting system* for algorithm $\mathcal{A}$ is a tuple $(\mathbb{R}^m, \mathbf{w}_\mathcal{A}, \xi_\mathcal{A})$. $\mathbb{R}^m$ is a vector space over the real numbers with dimension m. The function $\mathbf{w}_\mathcal{A} : (0,1] \mapsto \mathbb{R}^m$ is called the *weighting function*. For each $j \le n$, $\mathbf{w}_\mathcal{A}(x)$ is constant for $x \in I_j$. The function $\xi_\mathcal{A} : \mathbb{R}^m \mapsto \mathbb{R}$ is called the *consolidation function*. We have

$$\xi_\mathcal{A}(\mathbf{x}) = \xi^j_\mathcal{A}(\mathbf{x}), \qquad \text{if } \mathbf{x} \in D_j$$

for some set $\xi^1_\mathcal{A}, \ldots, \xi^\Lambda_\mathcal{A}$ of linear functions and some set $D_1, \ldots, D_\Lambda$ of disjoint domains covering $\mathbb{R}^m$. We require that $\xi_\mathcal{A}$ is continuous and has the *scalability* property: $\xi_\mathcal{A}(a\,\mathbf{x}) = a\,\xi_\mathcal{A}(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^m$ and $a \in \mathbb{R}$. Since $\xi_\mathcal{A}$ is continuous, and each piece is linear, the boundaries defining each domain are defined by at most $\Lambda - 1$ linear functions. I.e. each domain can be described using at most $\lambda_i < \Lambda$ constraints:

$$\mathbf{x} \in D_i \quad \Rightarrow \quad \begin{array}{r} \mathbf{x} \cdot \mathbf{d}_{i,1} \ge 0 \\ \mathbf{x} \cdot \mathbf{d}_{i,2} \ge 0 \\ \vdots \\ \mathbf{x} \cdot \mathbf{d}_{i,\lambda_i} \ge 0. \end{array}$$

Finally, for $(\mathbb{R}^m, \mathbf{w}_\mathcal{A}, \xi_\mathcal{A})$ to be a weighting system we must have

$$\text{cost}_\mathcal{A}(\sigma) \le \xi_\mathcal{A}\left(\sum_{i=1}^{N} \mathbf{w}_\mathcal{A}(p_i)\right) + O(1),$$

for all input sequences σ. Intuitively, in the simplest case, the weight of a piece indicates the maximum portion of a bin that it can occupy.

Weighting systems can be used to analyze Harmonic, Refined Harmonic, Modified Harmonic, Modified Harmonic 2, Harmonic+1 and Harmonic++. In fact, all these algorithms are instances of Super Harmonic. We develop a general analysis of Super Harmonic and apply this analysis to prove upper bounds for these specific algorithms.

We define a $2K+1$ dimensional weighting system for Super Harmonic. In order to express the vectors in compact format, we define the unit basis vectors: $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_K, \mathbf{r}_1, \ldots, \mathbf{r}_K$. The weighting function is

$$\mathbf{w}_{\text{SH}}(x) = \begin{cases} (1-\alpha_i)\dfrac{\mathbf{b}_{\phi(i)}}{\beta_i} + \alpha_i \dfrac{\mathbf{r}_{\varphi(i)}}{\gamma_i} & \text{if } x \in I_i \text{ with } i \le n, \\ x\dfrac{\mathbf{b}_0}{1-\epsilon} & \text{if } x \in I_{n+1}. \end{cases}$$

The consolidation function is

$$\xi_{\mathrm{SH}}(\mathbf{x}) = \mathbf{x}\cdot\mathbf{b}_0 + \max_{1\le k\le K+1} \min\left\{ \sum_{i=k}^{K} \mathbf{x}\cdot\mathbf{r}_i + \sum_{i=1}^{K} \mathbf{x}\cdot\mathbf{b}_i,\ \sum_{i=1}^{K} \mathbf{x}\cdot\mathbf{r}_i + \sum_{i=1}^{k-1} \mathbf{x}\cdot\mathbf{b}_i \right\}.$$

Lemma 4. *For all* σ, $\mathrm{cost}_{\mathrm{SH}}(\sigma) \le \xi_{\mathrm{SH}}\left(\sum_{i=1}^{N} \mathbf{w}_{\mathrm{SH}}(p_i)\right) + O(1)$.

Due to space considerations, the proof is given in the appendix.

4 General Analysis with Weighting Systems

We now turn our attention to analyzing algorithms. We begin by developing general techniques applicable to the analysis of any interval classification algorithm—we show how weighting systems can be used to upper bound the asymptotic performance ratio of a given algorithm. We then focus our analysis on Super Harmonic.

Suppose we have an online interval classification algorithm $\mathcal{A}$, with weighting system $(\mathbb{R}^m, w_{\mathcal{A}}, \xi_{\mathcal{A}})$. Fix an input σ.

Consider the optimal offline solution for σ. Suppose some bin in the optimal solution is not full. Let x be the sum sizes of the pieces in this bin. Then add a piece of size $1 - x$ to the end of our sequence. The cost of the optimal solution does not increase, whereas the cost to $\mathcal{A}$ cannot decrease. Therefore, when upper bounding the performance ratio of $\mathcal{A}$, we may assume that each bin in the optimal solution is full.

A *pattern* is a tuple $q = \langle q_1, \ldots, q_n \rangle$ over $\mathbb{N}$ such that $\sum_{i=1}^{n} q_i\, t_{i+1} < 1$. Intuitively, a pattern describes the contents of a bin in the optimal offline solution. The reader should contrast this with the definition of a packing given earlier. The weight of pattern q is

$$\mathbf{w}_{\mathcal{A}}(q) = \mathbf{w}_{\mathcal{A}}\left(1 - \sum_{i=1}^{n} q_i t_{i+1}\right) + \sum_{i=1}^{n} q_i\, \mathbf{w}_{\mathcal{A}}(t_i).$$

Define $\mathcal{Q}$ to be the set of all patterns q. Note that $\mathcal{Q}$ is necessarily finite.

A *distribution* is a function $\chi : \mathcal{Q} \mapsto \mathbb{R}_{\ge 0}$ such that $\sum_{q\in\mathcal{Q}} \chi(q) = 1$. Given σ, $\mathcal{A}$ is defined by the numbers and types of items it places in each of the bins it uses. Specifically, $\mathcal{A}$ is defined by a distribution χ. It uses $\mathrm{cost}(\sigma)\chi(q)$ bins containing items as described by the pattern q.

To show that $\mathcal{A}$ has performance ratio at most c, we show that

$$\begin{aligned}\frac{1}{\mathrm{cost}(\sigma)}\,\xi_{\mathcal{A}}\left(\sum_{i=1}^{N} \mathbf{w}_{\mathcal{A}}(p_i)\right) &= \frac{1}{\mathrm{cost}(\sigma)}\,\xi_{\mathcal{A}}\left(\sum_{q\in\mathcal{Q}} \mathrm{cost}(\sigma)\chi(q)\mathbf{w}_{\mathcal{A}}(q)\right)\\ &= \xi_{\mathcal{A}}\left(\sum_{q\in\mathcal{Q}} \chi(q)\mathbf{w}_{\mathcal{A}}(q)\right).\end{aligned}$$

is at most c, for all σ. The second step follows from the scalability of $\xi_{\mathcal{A}}$.

We are therefore led to consider the following optimization problem: Maximize $\xi^i_{\mathcal{A}}(\mathbf{x})$ subject to

$$\mathbf{x} = \sum_{q \in \mathcal{Q}} \chi(q) \mathbf{w}_{\mathcal{A}}(q); \tag{1}$$

$$0 \le \mathbf{x} \cdot \mathbf{d}_{i,j}, \qquad \text{for } 1 \le j \le \lambda_i; \tag{2}$$

$$0 \le \chi(q), \qquad \text{for } q \in \mathcal{Q}; \tag{3}$$

$$1 = \sum_{q \in \mathcal{Q}} \chi(q); \tag{4}$$

$$i \in \{1 \ldots \Lambda\}; \tag{5}$$

over integer variable i, real variables $\chi(q), q \in \mathcal{Q}$ and real vector $\mathbf{x}$. The value of this mathematical program, which we name $\mathcal{P}$, upper bounds the asymptotic performance ratio of $\mathcal{A}$. Fix $i = j$ and call the resulting linear program $\mathcal{P}_j$. We can solve $\mathcal{P}$ by solving P_j for $i = 1, \ldots, \Lambda$ and taking the maximum value. The following lemma tells us something about the structure of a solution to P_j:

Lemma 5. *For $1 \le j \le \Lambda$, there exists an optimal feasible solution to $\mathcal{P}_j$ where $\chi(q)$ is non-zero at at most $\lambda_i + 1$ patterns q.*

Proof. $\mathcal{P}_j$ is a $|\mathcal{Q}| - 1$ dimensional linear program, since $\mathbf{x}$ and (4) may be removed by substitution. By the theory of linear programming, the optimal solution is achieved at some vertex of the polytope of feasible solutions defined by (2) and (3). At a vertex of this polytope, $|\mathcal{Q}| - 1$ inequalities are satisfied with equality. Of these, at least $|\mathcal{Q}| - 1 - \lambda_i$ must be of the form (3), and each of these implies that some value of χ is zero. The total number of variables is $|\mathcal{Q}|$ and $|\mathcal{Q}| - (|\mathcal{Q}| - 1 - \lambda_i) = \lambda_i + 1$. □

For certain types of consolidation functions, stronger results are possible:

Lemma 6. *If $\xi_{\mathcal{A}}(\mathbf{x}) = \max_{1 \le j \le \Lambda} \xi^j_{\mathcal{A}}(\mathbf{x})$ for some set $\xi^1_{\mathcal{A}}, \ldots, \xi^\Lambda_{\mathcal{A}}$ of linear functions then there exits an optimal feasible solution to $\mathcal{P}$ where $\chi(q)$ is non-zero at at most one pattern q.*

Proof. Suppose distribution χ defines an optimal feasible solution. The objective value achieved is

$$\begin{aligned} \max_{1 \le j \le \Lambda} \xi^j_{\mathcal{A}} \left(\sum_{q \in \mathcal{Q}} \chi(q) \mathbf{w}_{\mathcal{A}}(q) \right) &= \sum_{q \in \mathcal{Q}} \chi(q) \xi^\ell_{\mathcal{A}}(\mathbf{w}_{\mathcal{A}}(q)) \\ &\le \max_{q \in \mathcal{Q}} \xi^\ell_{\mathcal{A}}(\mathbf{w}_{\mathcal{A}}(q)) = \xi^\ell_{\mathcal{A}}(\mathbf{w}_{\mathcal{A}}(q^*)). \end{aligned}$$

for some $1 \le \ell \le \Lambda$ and $q^* \in \mathcal{Q}$. The first step uses the linearity of $\xi^1_{\mathcal{A}}, \ldots, \xi^\Lambda_{\mathcal{A}}$, while the second uses the fact that χ is a convex combination over $\mathcal{Q}$. So the distribution

$$\chi'(q) = \begin{cases} 1 \text{ if } q = q^*, \\ 0 \text{ otherwise.} \end{cases}$$

achieves at least the objective value achieved by χ. We need to show that χ is feasible for $i = \ell$. Let $\mathbf{x}' = \sum_{q \in \mathcal{Q}} \chi'(q)\mathbf{w}_{\mathcal{A}}(q) = \mathbf{w}_{\mathcal{A}}(q^*)$. If χ' is feasible for $i = \ell$ we have $\xi^{\ell}_{\mathcal{A}}(\mathbf{x}') \geq \xi^{j}_{\mathcal{A}}(\mathbf{x}')$ for all j. If $\xi^{j}_{\mathcal{A}}(\mathbf{x}') > \xi^{\ell}_{\mathcal{A}}(\mathbf{x})$ for some $j \neq \ell$, this would contradict the optimality of χ. □

Note that the preceding lemma is applicable in the analysis of Harmonic, Refined Harmonic and Modified Harmonic, but not Harmonic+1. [8] and [10] use it implicitly. As we shall see, Richey [11] uses it incorrectly. Using Lemma 6, our problem is reduced to that of finding the single pattern q which maximizes $\xi_{\mathcal{A}}(\mathbf{w}_{\mathcal{A}}(q))$. This leads us to consider the mathematical program: Maximize $\xi^{i}_{\mathcal{A}}(\mathbf{x})$ subject to

$$\mathbf{x} = \mathbf{w}_{\mathcal{A}}(y) + \sum_{j=1}^{n} q_j \mathbf{w}_{\mathcal{A}}(t_j); \tag{6}$$

$$y = 1 - \sum_{j=1}^{n} q_j \, t_{j+1} \tag{7}$$

$$0 \leq \mathbf{x} \cdot \mathbf{d}_{i,j}, \qquad \text{for } 1 \leq j \leq \lambda_i; \tag{8}$$

$$y > 0, \tag{9}$$

$$q_j \in \mathbb{N}, \qquad \text{for } 1 \leq j \leq n, \tag{10}$$

$$i \in \{1 \ldots \Lambda\}; \tag{11}$$

over variables $\mathbf{x}, y, i, q_1, \ldots, q_n$. Intuitively, q is a pattern; q_j is the number of type j pieces in q. y is an upper bound on space space available for type $n+1$ pieces. Note that strict inequality is required in (9) because a type j piece is strictly larger than t_{j+1}. Call this integer linear program $\bar{\mathcal{P}}$. The value of $\bar{\mathcal{P}}$ upper bounds the asymptotic performance ratio when the consolidation function satisfies the conditions of Lemma 6. One can think of $\bar{\mathcal{P}}$ as a knapsack problem: The adversary must pack items into a knapsack of size 1. The profit for a knapsack is found by applying the consolidation function to the sum of the weights.

The following lemma allows for further simplification in the case that the algorithm under consideration uses Harmonic to pack items below a certain size:

Lemma 7. *Let ℓ and $k \leq \ell$ be positive integers, and $y \leq 1/k$ be a positive real number. The mathematical program: Maximize*

$$\sum_{i=1}^{\ell-k} \frac{v_i}{k+i-1} + \frac{\ell}{\ell-1}(y - z)$$

subject to

$$z < y;$$

$$z = \sum_{i=1}^{\ell-k} \frac{v_i}{k+i};$$

$$q_i \in \mathbb{N}, \qquad \textit{for } 1 \leq i \leq \ell - k;$$

over variables $v_1, \ldots, v_{\ell-k}$ and z has value $\Gamma(y, k, \ell)$ where

$$\Gamma(x,i,j) = \begin{cases} \dfrac{j}{j-1}x & \text{if } i = j \\ \dfrac{1}{i} + \Gamma\left(x - \frac{1}{i+1}, i+1, j\right) & \text{if } x > \frac{1}{i+1} \\ \Gamma(x,i+1,j) & \text{otherwise.} \end{cases}$$

Due to space considerations, the proof is given in the appendix.

Note that the lemma implies that HARMONIC has performance ratio $\Gamma(1,1,n+1)$. Further, it is easily seen that $\lim_{n\to\infty}\Gamma(1,1,n+1) = h_\infty$.

Using the machinery we have developed here, it is also easy to analyze the performance ratios of REFINED HARMONIC and MODIFIED HARMONIC. One merely need evaluate $\bar{\mathcal{P}}$; this is easily accomplished even by hand calculation.

We now turn to SUPER HARMONIC. We abuse notation and define $\mathbf{w}_i = \mathbf{w}_{\mathrm{SH}}(t_i)$ for the remainder of our discussion. Fix a $k \in \{1,\ldots,K+1\}$ and define

$$\mathbf{s} = \mathbf{b}_0 + \sum_{i=k}^{K}\mathbf{r}_i + \sum_{i=1}^{K}\mathbf{b}_i, \qquad \mathbf{t} = \mathbf{b}_0 + \sum_{i=1}^{K}\mathbf{r}_i + \sum_{i=1}^{k-1}\mathbf{b}_i.$$

First note that $\min\{\mathbf{x}\cdot\mathbf{s}, \mathbf{x}\cdot\mathbf{t}\} = \mathbf{x}\cdot\mathbf{t}$ for $k = 1$ and $\min\{\mathbf{x}\cdot\mathbf{s}, \mathbf{x}\cdot\mathbf{t}\} = \mathbf{x}\cdot\mathbf{s}$ for $k = K+1$. In these two cases, we can apply Lemma 6, and the performance ratio is upper bounded by the value of $\bar{\mathcal{P}}$.

We now turn to $2 \le k \le K$. Consider the mathematical program: Maximize

$$\min\{(z\,\mathbf{x} + (1-z)\mathbf{x}')\cdot\mathbf{s},\ (z\,\mathbf{x} + (1-z)\mathbf{x}')\cdot\mathbf{t}\}$$

subject to

$$\begin{array}{lll} z \in [0,1]; & & \\ \mathbf{x} = \dfrac{1}{1-\epsilon}\,y\,\mathbf{b}_0 + \displaystyle\sum_{j=1}^{n} q_j\mathbf{w}_j; & & \mathbf{x}' = \dfrac{1}{1-\epsilon}\,y'\,\mathbf{b}_0 + \displaystyle\sum_{j=1}^{n} q'_j\mathbf{w}_j; \\ y = 1 - \displaystyle\sum_{j=1}^{n} q_j\,t_{j+1}; & & y' = 1 - \displaystyle\sum_{j=1}^{n} q'_j\,t_{j+1}; \\ y > 0; & & y' > 0; \\ q_j \in \mathbb{N}, & & q'_j \in \mathbb{N}, \qquad \text{for } 1 \le j \le n; \end{array}$$

over variables $z, \mathbf{x}, \mathbf{x}', y, y', q_1,\ldots,q_n, q'_1,\ldots,q'_n$. Call this program $\hat{\mathcal{P}}_k$. By Lemma 5, if we show that value of $\hat{\mathcal{P}}_k$ is at most c for all k, the asymptotic performance ratio of SUPER HARMONIC is at most c. The two patterns are q and q', their weights are $\mathbf{x}$ and $\mathbf{x}'$, respectively. The constraints guarantee that q and q' are legitimate patterns. Conversely, the reader should verify that the constraints allow all possible patterns. The distribution between q and q' is given by z. Again, $\hat{\mathcal{P}}_k$ is a type of knapsack problem: The adversary must pack items into two knapsacks of size 1. The profit is found by applying the consolidation function a convex combination of the weights in the two knapsacks.

We have developed a branch and bound algorithm for solving $\hat{\mathcal{P}}_k$. We implemented this algorithm in *C++*. To ensure the validity of our results, all calculations are done using the GNU CLN infinite precision rational arithmetic package. A front end program is written in *Mathematica*. An explanation of the algorithm along with a listing of the code appears in the appendix. The program is available on the World Wide Web at

`http://www.csc.lsu.edu/~seiden/super_harmonic.tgz`

5 Results

Details of Harmonic++ are given in the appendix. Using the methods outlined in the preceding sections, we are able to show our main results:

Theorem 1. *The asymptotic performance ratio of* Harmonic++ *is at most* 158889/100000.

Theorem 2. *The asymptotic performance ratio of* Harmonic+1 *is at least* 1.59217.

Due to space considerations, the proofs is given in the appendix.

Using our computer program, we have also verified the upper bounds for Harmonic, Refined Harmonic, Modified Harmonic and Modified Harmonic 2.

6 Conclusions

We have developed a uniform method of analysis for online bin packing algorithms. Using our framework, we have analyzed the Super Harmonic algorithm. Online bin packing algorithms based on Harmonic are just special instances of this general algorithm. We have developed an instance of Super Harmonic, called Harmonic++, which has the best performance of any online bin packing algorithm to date. Our framework is easily adapted to closely related problems, such as variable-sized bin packing [12] and resource augmented bin packing [5].

The question of how to design a Super Harmonic algorithm is still an open one. The problem of choosing interval breakpoints is at the heart of this question. Once this choice is made, the values α_i can be optimized by mathematical (in some cases linear) programming. The solution to the breakpoint choice problem seems to be currently out of our reach. The results of Ramanan *et al.* [10] imply that any instance of Super Harmonic has performance ratio at least $19/12 > 1.58333$. Further, the set of intervals designed by Richey works very well, despite the ad-hoc design. We did not find another set which would significantly improve performance. An understanding of the breakpoint problem will not lead to a large improvement in performance, or bring us close to the lower bound of 1.54014. Still, despite the inherent limitations of our approach, it is our hope that this work brings us one step closer to a final resolution of the online bin packing problem.

Acknowledgement. We would like to thank Gerhard Woeginger for suggesting this problem.

References

1. Brown, D. J. A lower bound for on-line one-dimensional bin packing algorithms. Tech. Rep. R-864, Coordinated Sci. Lab., University of Illinois at Urbana-Champaign, 1979.
2. Coffman, E. G., Garey, M. R., and Johnson, D. S. Approximation algorithms for bin packing: A survey. In *Approximation Algorithms for NP-hard Problems*, D. Hochbuam, Ed. PWS Publishing Company, 1997, ch. 2.
3. Csirik, J. An on-line algorithm for variable-sized bin packing. *Acta Informatica 26*, 8 (1989), 697–709.
4. Csirik, J., and Woeginger, G. On-line packing and covering problems. In *On-Line Algorithms—The State of the Art*, A. Fiat and G. Woeginger, Eds., Lecture Notes in Computer Science. Springer-Verlag, 1998, ch. 7.
5. Csirik, J., and Woeginger, G. Resource augmentation for online bounded space bin packing. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming* (Jul 2000), pp. 296–304.
6. Johnson, D. S. Fast algorithms for bin packing. *Journal Computer Systems Science 8* (1974), 272–314.
7. Johnson, D. S., Demers, A., Ullman, J. D., Garey, M. R., and Graham, R. L. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing 3* (1974), 256–278.
8. Lee, C., and Lee, D. A simple on-line bin-packing algorithm. *Journal of the ACM 32*, 3 (Jul 1985), 562–572.
9. Liang, F. M. A lower bound for online bin packing. *Information Processing Letters 10* (1980), 76–79.
10. Ramanan, P., Brown, D., Lee, C., and Lee, D. On-line bin packing in linear time. *Journal of Algorithms 10*, 3 (Sep 1989), 305–326.
11. Richey, M. B. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics 34* (1991), 203–227.
12. Seiden, S. S. An optimal online algorithm for bounded space variable-sized bin packing. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming* (Jul 2000), pp. 283–295.
13. van Vliet, A. An improved lower bound for online bin packing algorithms. *Information Processing Letters 43*, 5 (Oct 1992), 277–284.
14. Yao, A. C. C. New algorithms for bin packing. *Journal of the ACM 27* (1980), 207–227.

Quick k-Median, k-Center, and Facility Location for Sparse Graphs

Mikkel Thorup

AT&T Labs–Research, Shannon Laboratory, 180 Park Avenue, Florham Park, NJ 07932. mthorup@research.att.com

Abstract. Solving an open problem of Jain and Vazirani [FOCS'99], we present $\tilde{O}(n+m)$ time constant factor approximation algorithms for the k-median, k-center, and facility location problems with assignment costs being shortest path distances in a weighted undirected graph with n nodes and m edges.
For all of these location problems, $\tilde{O}(n^2)$ algorithms were already known, but here we are addressing large sparse graphs. An application could be placement of content distributing servers on the Internet. The Internet is large and changes so frequently that an $\tilde{O}(n^2)$ time solution would likely be outdated long before completion.

1 Introduction

We consider several classical (discrete) location problems defined in terms of a a metric (P, dist), $|P| = n$. We want to pick a set $S \subseteq P$ of *facilities* subject to different objectives. In the *k-median* and *k-center problems* we require $|S| = k$, and our goal is to minimize $\sum_{x\in P} \text{dist}(x, S)$ and $\max_{x\in P} \text{dist}(x, S)$, respectively. In the *facility location problem*, there is no limit k on the number of facilities, but we are further given a facility cost function f-cost : $P \to \mathbb{N}_0$, and our goal is to minimize $\sum_{a\in S} \text{f-cost}(a) + \sum_{x\in P} \text{dist}(x, S)$.

In this paper, we are interested in the *graph setting* where the metric is the shortest path metric of a weighted undirected connected graph $G = (V, E, \ell : E \to \mathbb{N})$, $|V| = n, |E| = m \geq n-1$, that is, $\text{dist}(x, y)$ is the length of the shortest path from x to y in G. This setting used is the basis for many of the classical applications of facility location in operations research [16]. For example, the problems can then model placement of shopping centers on a road network with driving distance to nearest shopping center the consumer cost. Also, they can model placement of content distribution servers on the Internet. Both examples concern large sparse graphs. Further, the Internet changes frequently and hence an algorithm has to be fast in order to produce up-to-date answers. In particular, we cannot wait for months for an all-pairs shortest paths computation to finish.

For all of the above location problems, we present $\tilde{O}(m)$ time constant factor approximation algorithms. Here $\tilde{}$ means that we ignore $\log n$ factors. The concrete approximation factors are $12 + o(1)$ for k-median, 2 for k-center, $3 + o(1)$ for facility location. The approximation factor for the k-median problem may

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 249–260, 2001.

be reduced to around 9, but this is complicated and beyond the scope of the current paper.

Our results solve an open problem of Jain and Vazirani [13]. They considered a *distance oracle* setting where given any pair of points $(x, y) \in P^2$, one can compute $\text{dist}(x, y)$ in constant time. For the k-median and facility location problems, the achieved approximation factors of 6 and 3, respectively, in $\tilde{O}(n^2)$ time[1], improving for the k-median, the LP-based factor $6\frac{2}{3}$ from [1]. They noted that "The distinguishing feature of our algorithms is their low running time". In their final discussion they ask if improved running times can be obtained for the graph version in the case of sparse graphs, as done near-optimally in this paper. We note that the distance oracle setting is interesting in its own right, e.g. interpreting the location problems as clustering problems for large data bases [7]. As it turns out, a by-product of our work is some much improved approximation factors in "sub-linear" $o(n^2)$ time in the distance oracle setting.

It should be appreciated that the graph and distance oracle models are fundamentally different in that a single distance query in a graph takes $O(m)$ time [17]. For a sparse graph, Jain and Vazirani would first compute the shortest path metric with an *all pairs* shortest path computation, which takes $O(nm)$ time. Our improvement to $\tilde{O}(m)$ time is obtained using only a polylogarithmic number of *single source* shortest path computations. We note that it is easy to get an $\tilde{O}(n^2)$ time solution using approximate shortest path computations (see e.g. [3]), but the approximation factor become worse than ours. More importantly, as suggested in [13] and as in our example applications, we are mostly interested in large sparse graphs with $m = O(n)$, and then our $\tilde{O}(m)$ time is a strong improvement.

The pride of this paper is the solution to the k-median problem, the hardness being the sharp bound on the number of facilities. Facility location is comparatively trivial because we can use approximate facility costs. It is considered here for completeness because it was part Jain and Vazirani's open problem [13]. The k-center solution comes in for free as a warm-up for our solution to the facility location problem. Also, by covering k-median, k-center, and facility location, we develop a quite general tool-box for basic location problems in networks [16].

Henceforth, the paper is focused on the k-median problem, leaving k-center and facility location to two independent sections at the end (c.f. §5 and §6).

The distance oracle version. Our work on the graph version of the k-median problem is inspired by the progress on the distance oracle version: Indyk [10] has presented a randomized reduction that together with the $\tilde{O}(fn)$ time factor 6 approximation algorithm of Jain and Vazirani [13] implies a $\tilde{O}(k^3 n)$ time factor $(3 + o(1))(2 + 6) = 24 + o(1)$ approximation algorithm, though cheating a bit using $2k$ facilities. Also, as part of their work on an on-line version of the k-median problem, Mettu and Plaxton [15] presented an $O(n^2)$ time algorithm with an approximation factor slightly below 40. Finally, based on Indyk's construction [10],

[1] Actually, they get $\tilde{O}(fn)$ time if only f points are potential facilities, but here we generally assume all points are potential facilities.

Guha et al. [7] presented a $\tilde{O}(kn)$ time factor $6 \times 2 \times (24 + o(1) + 1) = 300 + o(1)$ approximation algorithm. Their algorithm works for a special streaming version of the problem if distances between points can be computed directly from the points. Their approximation factor can be reduced to $80 + o(1)$ if $k = \tilde{O}(\sqrt{n})$ [Guha, personal communication]. An $\tilde{O}(kn)$ time constant factor approximation is also announced by Mettu and Plaxton [15], but with no specification of the constant. They also pointed out that $\Omega(kn)$ time is necessary even for randomized algorithms.

Some of our initial developments actually imply an $\tilde{O}(kn)$ time factor $12+o(1)$ approximation for the distance oracle version of the k-median problem. This is better than any previous $o(n^2)$ time algorithm, and 25 times better than the previous $\tilde{O}(kn)$ time algorithm [7] for unbounded k. Clearly some of the previous approximation factors could have been improved somewhat, but our vast improvement is due to some new simple and powerful sampling techniques. Corresponding improvments can also be obtained in the streaming model considered in [7] (historically this paper actually predates [7] slightly, but had a less fortunate conference history).

The graph version. The previous work on the distance oracle version does have some applications for the graph version. Applying an all pairs small-stretch path algorithm of Cohen and Zwick [3], we can approximate all distances in the graph within a factor 3 in $\tilde{O}(n^2)$ time, and then we can apply Jain and Vazirani's algorithm [13] in $\tilde{O}(n^2)$ time to get a factor $3 \times 6 = 18$ approximation algorithm. In order to exploit the $\tilde{O}(kn)$ time distance oracle algorithm of Guha et al. [7], we can first apply Thorup and Zwick's [19] approximate distance oracle algorithm: for any positive integer t, after $O(tmn^{1/t})$ preprocessing time, we can answer distance queries within a factor $2t-1$ in $O(t)$ time. Setting $t = 2$ and combining with [7], we get an $\tilde{O}(m\sqrt{n} + nk)$ time algorithm with approximation factor $3(300 + o(1)) = 900 + o(1)$, or $3(80 + o(1)) = 240 + o(1)$ if $k = \sqrt{n}$.

Our new $\tilde{O}(m)$ time bound is near-optimal and breaks the $\Omega(kn)$ lower bound for distance oracles if $k \gg m/n$. Further, our approximation factor is only $12 + o(1)$. This is better than any previous $o(nm)$ time solution. It appears the approximation factor can be further reduced to around 9.

Our approach is easily generalized to work for weighted points. One application of this is if only a subset of the points are potential facilities. Then we first assign each point to its nearest potential facility. Second we solve the k-median problem over the potential facilities, each weighted by the number of assigned original points. The resulting solution can be seen to be a factor $2(12 + o(1)) + 1 = 25 + o(1)$ approximation.

Other metrics. Our techniques imply that the k-median problem can be solved with $\tilde{O}(n)$ nearest neighbor queries. For Hamming space this implies a constant factor approximation in $\tilde{O}(n^{1+\varepsilon})$ time for $\varepsilon > 0$ [11], thus beating that $\Omega(kn)$ lower bound for general distance oracles if $k = n^{\Omega(1)}$. We note that large values of k are relevant to fine grained clustering.

(Im)practicality and outline. Our $\tilde{O}(m)$ solution to the k-median problem in graphs is exceedingly complicated, and hence unlikely to be of practical relevance. However, in our developments we present several algorithms that are simple and easy to implement, yet providing stronger bounds than were previously known. Also, our algorithms for the k-center and facility location problems are simple and easy to implement, and it is all these simpler algorithms that constitute the practical contribution of the paper.

First, in §2, we present a simple fast randomized algorithm for selecting a set of $\tilde{O}(k)$ potential facilities, guaranteed to contain a solution with k facilities with at most twice the cost of the optimal solution. Using this as a preprocessing step for Jain and Vazirani's algorithm, we find a factor $12 + o(1)$ approximation in $\tilde{O}(kn)$ time for distance oracles, and $\tilde{O}(km)$ time for graphs. This part is considered very practical, "perfect" for distance oracles, and good for graphs if k is not too large.

Next, in §3, we show that it can be meaningful to apply the algorithm from [13] to a sparse graph whose weights do not satisfy the triangle inequality, and construct a graph for which this makes sense. In $\tilde{O}(m)$ time, this leads to a factor $12+o(1)$ approximation to the k-median problem but cheating using $k+k/\log^2 n$ facilities. This part is still simple enough to be of practical use, and good enough if the bound on the number of facilities is not sharp.

The true difficulty in the k-median problem is the sharp bound on the facilities. In §4, we sketch a very convoluted recursion for getting rid of the last $k/\log^2 n$ extra facilities, leaving the details to the journal version. It is based on structural theorems showing that if we cannot easily get rid of the $k/\log^2 n$ extra facilities, it is because our current solution is very similar to any optimal solution. This similarity allows us to fix most of the facilities and recurse on a $o(k)$-median problem. This last step is considered too complicated to be of practical interest, but it takes us to our theoretical goal: a near-linear time constant factor approximation to the k-median problem.

Notation and terminology. We are dealing with a metric (P, dist), $|P| = n$, from which we pick a subset S of facilities. Further, each point $x \in P$ is assigned a facility $x^S \in S$ at cost $\text{dist}(x, x^S)$. The total cost of S is then $\text{cost}(S) = \sum_{x\in P} \text{dist}(x, x^S)$ Unless otherwise stated, we assume that x^S is a point in S nearest to x. Then the assignment cost for x is $\text{dist}(x, S) = \min_{a\in S} \text{dist}(x, a)$, and then the total cost of S is $\text{cost}(S) = \sum_{x\in P} \text{dist}(x, S)$.

By evaluating a set $S \subseteq P$, we mean that for each $x \in P$, we find its nearest point x^S in S, compute $\text{dist}(x, x^S) = \text{dist}(x, S)$, and finally, compute $\text{cost}(S)$.

Observation 1. *In an undirected (or directed) weighted graph, we can evaluate a set S in $O(m)$ time ($O(m \log\log n)$ time).*

Proof. Introduce a source s with zero length edges to all $a \in S$, and compute the shortest path tree to all nodes in $O(m)$ time [17]. For each x, $\text{dist}(x, S)$ is the found distance from s, and $\text{cost}(S)$ is the sum of these distances. Finally, for each x in the subtree of $a \in S$, we set $x^S = a$. The same construction works for directed graphs if we first reverse the direction of all edges and use the directed shortest path algorithm from [18].

Note that if for some reason, we want to run on a restricted machine model such as the pointer machine, we get an $O(n \log n + m)$ time bound in Observation 1 from [4].

The *k-median problem* is the problem of finding $S \subseteq P$, $|S| \leq k$, minimizing $\text{cost}(S)$. Define $k\text{-mediancost}^{\subseteq F} = \min\{\text{cost}(S) : S \subseteq F, |S| = k\}$ and $k\text{-mediancost} = k\text{-mediancost}^{\subseteq P}$. By a c-approximation to the k-median problem, we mean a solution $S \subseteq P$, $|S| = k$, with $\text{cost}(S) \leq c \times k\text{-mediancost}$.

By the S-cluster of $a \in S$, we mean the set $\{x \in P : x^S = a\}$, denoted $\text{cluster}(a, S)$.

Generally, we will use a subscript to denote that measurements are done in a metric different from the one currently understood. For example, if H is a graph, $\text{dist}_H(x, y)$ is the distance from x to y in H.

2 Sampling $k \log^{O(1)} n$ Facilities

In this section, we will prove

Theorem 1. *For $0 < \varepsilon < 0.4$, with probability at least $1/2$, by sampling and evaluating $O(\log n)$ sets of size $O(k/\varepsilon \log n)$, we can identify a set F of size $O(k/\varepsilon \log^2 n)$ such that $k\text{-mediancost}^{\subseteq F} \leq (2+\varepsilon) \times k\text{-mediancost}$.*

Our constructive proof of Theorem 1 is based on the following simple algorithm:

Algorithm A

A.1. $R := P$; $S := \emptyset$;

A.2. while $|R| \geq k/\varepsilon \log^2 n$ do

A.2.1. add $4k/\varepsilon \log n$ random points from R to S

A.2.2. pick a random $t \in \{1, ..., |R|\}$ and remove from R the t points with lowest distance to S.

A.3. return $F = S \cup R$.

Proof (of Theorem 1). First we note that the probability that Algorithm A terminates in $\omega(\log n)$ iterations is $1/n^{\omega(1)}$ (each round reduces R by a factor 2 with probability $1/2$ and we can only do this $\log_2 n$ times) so we may assume termination in $O(\log n)$ iterations.

Let OPT be an optimal solution to the k-median problem. Our claimed good solution inside F will be $OPT^F = \{a^F\}_{a \in OPT}$. It is trivially of size k, and for our analysis, we will assign each $x \in P$ to $(x^{OPT})^F$.

A point $x \in P$ will be declared "happy" as soon as a point a with $\text{dist}(a, x^{OPT}) \leq \text{dist}(x, x^{OPT})$ is picked for S, to later end up in F. Clearly, if all points ended up happy, we would get $\text{cost}(OPT^F) \leq 2\text{cost}(OPT)$. Unfortunately, we cannot hope to make all points happy, but we will find a way to pay for the unhappy ones.

Consider an unhappy point x. Our assignment cost for x is $\text{dist}(x, (x^{OPT})^F) \leq \text{dist}(x, x^{OPT}) + \text{dist}(x^{OPT}, (x^{OPT})^F) \leq \text{dist}(x, x^{OPT}) + \text{dist}(x^{OPT}, x^F) \leq \text{dist}(x, x^{OPT}) + \text{dist}(x^{OPT}, x) + \text{dist}(x, x^F) \leq$

$2\text{dist}(x, OPT) + \text{dist}(x, F)$ The point x can itself pay $2\text{dist}(x, OPT)$, so it only remains to show that the sum over $\text{dist}(x, F)$ over all unhappy x is bounded by $\varepsilon\,\text{cost}(OPT)$.

Consider an iteration of the algorithm. Let S and R be the values of S and R after step A.2.1, and let U be the prefix of R removed in step A.2.2. We will show that the expected fraction of unhappy points in R and U is very small and based on this we will show that the happy points can pay for them.

Claim 1. *The expected fraction of unhappy points in R after step A.2.1 is $\leq \varepsilon/(4\log n)$.*

Proof. Consider a point $x \in R$ which was not happy before step A.2.1, and let C be the remaining part of the OPT-cluster containing x, that is, $C = \text{cluster}(x^{OPT}, OPT) \cap R$.

Suppose there are i points in C, including x, that are as close to x^{OPT} as x. Now, the probability that x is not turned happy by step A.2.1 is $(1 - i/|R|)^{4k/\varepsilon \log n} < e^{-i4k/\varepsilon \log n/|R|}$. Thus, no matter the size of C, the expected number of unhappy points in C is at most $\sum_{i=1}^{\infty} e^{-i4k/\varepsilon \log n/|R|} < \int_{x=0}^{\infty} e^{-x4k/\varepsilon \log n/|R|} dx < |R|/(4k/\varepsilon \log n)$, so with k clusters, the expected fraction of unhappy elements in R is at most $1/(4/\varepsilon \log n)$.

Claim 2. *If f is the fraction of unhappy points before step A.2.2 removes the random prefix U of R, the probability that*

$$\sum_{\text{unhappy } x \in U} \text{dist}(x, S) \leq \varepsilon \sum_{\text{happy } x \in U} \text{dist}(x, S)/2 \tag{1}$$

is not satisfied is at most $f(2 + 2/\varepsilon)$.

Proof. Let the points in R be sorted in order of increasing distance to S. We are going to delete the first t points in this sequence with t randomly chosen from $\{1, ..., |R|\}$. Before this happens, traverse R and let each unhappy point grab the first $\lceil 2/\varepsilon \rceil$ happy points of higher distance that are not grabbed yet, if any. Now, if point t is neither unhappy nor grabbed, then each unhappy point among the first t points have grabbed its own $\lceil 2/\varepsilon \rceil$ happy points to pay for it, and hence (1) is satisfied. The fraction of points that are unhappy or grabbed by an unhappy point is $f(\lceil 2/\varepsilon \rceil + 1)$.

By Claim 1 and 2, the probability that (1) is not satisfied for a given iteration is $E(f)(2+2/\varepsilon) \leq (1+\varepsilon)/(2\log n)$. Since the expected number of iterations is H_n, the probability that (1) is false for any iteration is $\leq (1+\varepsilon)H_n/(2\log n) < 1/2$ for $\varepsilon \leq 0.4$ and $n \to \infty$. Thus, we may assume that (1) is satisfied over all iterations.

Since $\text{dist}(x, S) \geq \text{dist}(x, F)$ for any unhappy x, and $\text{dist}(y, S) \leq 2\text{dist}(y, OPT)$ for any happy y, (1) implies $\sum_{\text{unhappy } x \in U} \text{dist}(x, F) \leq \varepsilon \sum_{\text{happy } x \in U} \text{dist}(x, OPT)/2$ However, since the sets U from different iterations are disjoint, we get $\sum_{\text{unhappy } x \in P} \text{dist}(x, F) \leq \varepsilon \sum_{\text{happy } x \in P} \text{dist}(x, OPT)/2 \leq \varepsilon\text{cost}(OPT)$, as desired.

It should be noted that the above construction works even if OPT is allowed to use points not in P, and in this case, we cannot get below 2 (think of OPT being the center of a star and the center not being in P). Also, note that for each $a \in OPT$, all points in cluster(a, OPT) are assigned to the same facility in F. Hence the construction works even if the facilities have the same limited capacity.

Corollary 1. *For a general metric we can find a $(12 + o(1))$-approximation for the k-median problem, using $\tilde{O}(kn)$ distance queries and computation time.*

Proof. By the above theorem and lemma, we spend $\tilde{O}(kn)$ distance queries on finding a set of $\tilde{O}(k)$ relevant facility locations, and then the algorithm from [13] allows us to solve the k-median problem in $\tilde{O}(kn)$ time.

3 Reducing to $k + k/\log^2 n$ Facilities

In this section, we will show that we can get down from $k \log^{O(1)} n$ potential facilities to a solution S with $k + k/\log^2 n$ facilities. We wish to apply the techniques of Jain and Vazirani [13], but these techniques are only quoted as working for graphs satisfying the triangle inequality. More precisely, let F be the set of potential facilities. They assume all edges in $F \times P$, and that each edge $(a, x) \in F \times P$ has length $\ell(a, x) = \text{dist}(a, x)$. If $|F| = \log^{\omega(1)} n$, this is too much for our time bounds.

We will now discuss what happens when the algorithm from [13] is applied and G does not satisfy the triangle inequality. Define $\ell\text{-cost}(S) = \sum_{x \in P} \ell(x, x^S)$. Here $\ell(x, x^S) = \infty$ if (x, x^S) is not an edge in G. What the algorithm from [13] really finds is a k-median $S \subseteq F$ with $\text{cost}(S) \leq 6 \times k\text{-median-}\ell\text{-cost}^{\subseteq F}$. The point is that the dual variables providing the lower bound in [13] do not require triangle inequality. It is only used in bounding the gap between the primal solution and the dual variables.

Concerning speed, the algorithm in [13] works in $\tilde{O}(m)$ time for m edges, except for the rounding. We will modify the simple rounding from [13, §3.3] to make it efficient. The simple rounding from [13, §3.3] operates on to subsets A and B of F with $|A| < k$ and $|B| > k$. First they let each $a \in A$ pick its nearest un-picked $b \in B$. Afterwards, they pick $k - |A|$ random facilities among the remaining $|B| - |A|$ facilities in B. However, picking the nearest un-picked facility is not easy. Instead, we evaluate B once in $\tilde{O}(m)$ time. For each $a \in A$, this gives us the nearest facility $a^B \in B$. We now first pick $A^B = \{a^B : a \in A\}$ and second we pick a random subset of $B \setminus A^B$ of size $k - |A^B|$. To see that the analysis in [13, §3.3] still goes through, we note that the probability that $b \in B \setminus A^B$ is picked is $(k - |A^B|)/(|B| - |A^B|) \leq (k - |A|)/(|B| - |A|)$.

Since we do not use the improved rounding in [13, §3.5], the overall approximation factor is worsened by a factor $(1 + o(1))$. In conclusion, we get

Theorem 2. *In $\tilde{O}(m)$ time we can find $S \subseteq F$, $|S| = k$, with* $\text{cost}(S) \leq (6 + o(1)) \times k\text{-median-}\ell\text{-cost}^{\subseteq F}$.

To apply this theorem, we construct a graph G^ℓ with $\tilde{O}(n)$ edges (v,w), each with $\ell(v,w) = \text{dist}(v,w)$, and with k^ℓ-median-ℓ-cost$_{G^\ell}^{\subseteq F} = O(k\text{-mediancost}^{\subseteq F})$ where $k^\ell = k + k\log^2 n$. Applying Theorem 2 then gives us a k^ℓ-median $S \subseteq F$ with $\text{cost}(S) \le \text{cost}_{G^\ell}(S) = O(k\text{-mediancost}^{\subseteq F})$.

The construction of G^ℓ is rather simple based on F. Set $d = \log^3 n|F|/k = \log^{O(1)} n$. For each point $x \in P$, we include an edge to each of the d nearest neighbors in F.

Lemma 1. *The graph G^ℓ has $(k + k/\log^2 n)$-median $S \subseteq F$ with ℓ-cost$(S) \le k$-mediancost$^{\subseteq F}$.*

Proof. All we need is to show the existence of a set $D \subseteq F$ of size $k/\log^2 n$ which dominates in the sense that each x, x^D is one of its d nearest neighbors. We then set $S = OPT \cup D$ where OPT is the optimal solution. If x^{OPT} is one of the d nearest neighbors in F, $x^S = x^{OPT}$. Otherwise, $x^S = x^D$ and then $\ell(x, x^D) \le \text{dist}(x, OPT)$, so $\ell\text{-cost}(S) \le \text{cost}(OPT)$.

To show the existence of D, we just pick D randomly. For each $x \in P$, the probability that none of its d nearest neighbors are picked is $\le (1 - |D|/|F|)^d < e^{-\log n} < 1/n$, so there is a positive probability that this does not happen for any x.

For the construction of G^ℓ, we need

Lemma 2. *With high probability, using $O(d \log n)$ evaluations, we can find the d nearest neighbors in F to each point x.*

Proof. We pick each $a \in F$ with probability $1/(2d)$ for a set Q that we evaluate. For each $x \in P$ and each $i \le d$, the probability that x^Q is the ith nearest neighbor in F is $\ge (1 - 1/(2d))^{i-1}/2d \ge 1/(4d)$. Hence, in $O(d \log n)$ evaluations, we can find the d nearest neighbors of all x with high probability.

Theorem 3. *With probability $1 - O(1/n)$, in $\tilde{O}(m)$ time, we can construct a $k + k/\log^2 n$-median S with* $\text{cost}(S) \le (12 + o(1)) \times k$-mediancost

Proof. First using Theorem 1, we identify a set F of size $k\log^{O(1)} n$ with k-mediancost$^{\subseteq F} \le (2 + o(1))k$-mediancost. Then using the above lemmas we construct G^ℓ with $(k + k/\log^2 n)$-median-ℓ-cost$_{G^\ell} \le k$-mediancost$^{\subseteq F}$ and finally, we apply Theorem 2 to G^ℓ.

To get the low error probability, we repeat the above construction $O(\log n)$ times, returning the S minimizing $\text{cost}(S)$. Note that this could not be done for Theorem 1 because we cannot compute and compare k-mediancost$^{\subseteq F}$.

4 Recursing Down to k Facilities

Let OPT denote some optimal solution to the k-median problem. Our starting point is a solution S, as from Theorem 3, of satisfactory cost, that is, $\text{cost}(S) = O(OPT)$, but which uses $q = k/\log^2 n$ too many facilities. We will then first try greedily to remove q facilities from S. If this cannot be done easily at cost $o(S)$, we will be able to fix many universally good facilities and then recurse. This recursion is by far the hardest and most technical part of the paper, and a descent presentation would take about 8 pages in the current format. For space reasons, we defer this to the journal version.

5 k-Center

We want to pick $S \subseteq V$, $|S| = k$ minimizing $\max_{v \in V} \mathrm{dist}(v, S)$. A factor 2 approximation is classical [6,8], and best possible [9], but the natural algorithm takes $\tilde{O}(km)$ time. We get down to $\tilde{O}(m)$ time, and our methods will be reused for facility location.

The classical factor 2 approximation is the following greedy algorithm: guess the optimal distance d^* and then return any maximal dist-$2d^*$ independent set. Here, for any d, a subset $U \subseteq V$ is dist-d *independent* if no two vertices of U are within distance d of each other. We know we have an adequate value of d if it gives rise to $\geq k$ facilities whereas $d-1$ gives rise to $< k$ facilities, so d may be found with a binary search.

The obvious greedy way of finding a maximal dist-d independent set U is as follows: set $U = \emptyset$ and $W = V$. While $W \neq \emptyset$, add an arbitrary vertex $v \in W$ to U and remove all vertices within distance d from W. The complexity of this algorithm is $\tilde{O}(|U|m)$. Here we we will get down to near-linear time. To the best of our knowledge, no $o(mn)$ time algorithm existed for finding maximal dist-d independent sets.

We shall use the following result of Cohen [2]:

Lemma 3 (Cohen). *For any $W \subseteq V$, after $\tilde{O}(m)$ preprocessing, for any vertex v and $d \geq 0$, in $\tilde{O}(1)$ time, we can estimate within a factor $1 \pm o(1)$ the number of vertices from W within any distance d of v.*

Proposition 1. *We can find a maximal* dist-d *independent set of any set $W \subseteq V$ in $\tilde{O}(m)$ time.*

Proof. As in the above greedy algorithm, start by setting $U = \emptyset$. While $W \neq \emptyset$, do as follows. Using Lemma 3, compute $\delta = (1 \pm o(1)) \max_{v \in W} |N_{\leq d}(v) \cap W|$. Pick a random subset R of W of size $|W|/\delta$. Let $T = \{v \in R | N_{\leq d} \cap R = \{v\}\}$. Add T to U and remove all vertices from W within distance d from U.

To identify the set T, construct a family $\{R_i\}_{i \leq 2 \log_2 |R|}$ of subsets of R such that for all $v, w \in R$ there is a set R_i containing v but not w. These sets may be constructed by associating different $\log_2 |R|$ bit vectors to the vertices in R, and then characterizing each set by the value of a certain bit position. Now, $N_{\leq d}(v) \cap R = \{v\}$ if and only if $\mathrm{dist}(v, R_i) > d$ for each R_i not containing v.

The idea in the above construction is that it within $O(\log n)$ rounds reduces $\max_{v \in W} |N_{\leq d}(v) \cap W|$ by a constant factor. More precisely, we show that if for some vertex $v \in W$, $|N_{\leq d}(v) \cap W| \geq \delta/2$, the next round eliminates v with constant probability. Clearly, the condition implies that some vertex $u \in N_{\leq d}(v) \cap W$ will be picked for R with constant probability. Further, by choice of δ, $|N_{\leq d}(u) \cap W| \leq (1 + o(1))\delta$, so with constant probability, no other vertex from u will be picked for R. Hence some $u \in N_{\leq d}(v) \cap W$ will end in T, eliminating v from W, with constant probability.

We note that the above proof has some similarities with the randomized parallel independent set algorithm in [14]. However, the algorithm in [14] accesses all edges whereas we do not want to consider the $O(n^2)$ pairs of distance $\leq d$.

Corollary 2. *In $\tilde{O}(m)$ time, we can find a factor 2 approximation to the k-center problem.*

In [8] are mentioned several similar problems that can be improved with similar methods.

6 Facility Location

In the *facility location problem* for a metric (P, dist), there is a facility cost f-cost(x) associated with each $x \in P$, and then the cost of $S \subseteq P$ is $\sum_{f \in S} \text{f-cost}(f) + \sum_{x \in P} \mathrm{dist}(x, S)$. First we note that in the distance oracle version of the facility location problem, even if all facility costs are uniform, no constant factor approximation is possible with $o(n^2)$ queries, not even if we allow randomization. For a negative example, divide into clusters of size t with intra- and inter-cluster distance 0 and ∞, respectively. Then we expect to need $\Omega(n^2/t)$ queries for an approximation factor substantially below t. It follows that Mettu and Plaxton's [15] $O(n^2)$ time bound is optimal for facility location with distance oracles, even with uniform facility costs.

However, in [5], it is shown that Jain and Vazirani's algorithm [13] can be implemented with $\tilde{O}(n)$ nearest neighbor queries, leading to a more efficient solution in Hamming space [11]. We note that [5] does not give anything for the k-median problem as it is based on approximate counting, hence approximate payment of facilities, and then Jain and Vazirani's rounding trick does not work.

In a graph, we have no efficient way of supporting individual nearest neighbor queries, but what we can do for a subset $X \subseteq V$ of the vertices is for all points $v \in V$ to find their nearest neighbor v^X, that is, in graphs, we can solve the *all points nearest marked neighbor* problem in $O(m)$ time (c.f. proof of Observation 1). Essentially, we show below that facility location can be solved within a factor 3 from optimality, with a polylogarithmic number of solutions to the all points nearest neighbor problem. We note that whereas "phase 2" of Jain and Vazirani's algorithm [13] is trivial to implement with efficient individual nearest neighbor queries [5], it needs something like Proposition 1 for graphs.

Instead of using Jain and Vazirani's algorithm [13], we use the one of Mettu and Plaxton [15]. The factor 3 approximation algorithm of Mettu and Plaxton [15] for facility location is very simple and elegant. As the algorithm in [13], it has two phases.

Phase 1. for each $x \in P$, we find r_x such that $\mathrm{value}(x, r_x) = \text{f-cost}(x)$ where $\mathrm{value}(x, r) = \sum_{y \in P, \mathrm{dist}(x,y) \leq r} (r - \mathrm{dist}(x, y))$.

Phase 2. Starting with $S = \emptyset$, we visit $x \in P$ in order of increasing r_x, adding x to S if $\mathrm{dist}(x, S) > 2r_x$.

Lemma 4 ([15]). *The above set S is at most a factor 3 from the optimal solution to the facility location problem.*

For an efficient implementation of the above algorithm, let $\varepsilon > 0$ be such that 2 is an integral power of $(1 + \varepsilon)$ for some integer i. Increasing assignment just

a little, we will view all distances as rounded up to nearest integral power of $(1+\varepsilon)$.

Recall that in our later usage of the r_x, all we care about is which vertices are within distance $2r_x$ from x. Let i be such that $(1+\varepsilon)^i \le r_x \le (1+\varepsilon)^{i+1}$, since 2 is an integral power of $(1+\varepsilon)$, there are no rounded distances between $(1+\varepsilon)^i$ and $(1+\varepsilon)^{i+1}$. Thus, we can freely round r_x down to the nearest power of $(1+\varepsilon)$, even if this implies $\text{value}(x, r_x) \ll \text{f-cost}(x)$.

Algorithm B Implements phase 1 finding r_v for all $v \in V$.

B.1.for all $a \in V$, set $f_a = 0$ — $f_a = \text{value}(a, (1+\varepsilon)^i)$.

B.2.set $U = V$ — U are facilities with unidentified r_a.

B.3.for $i = 1, 2, \ldots$ while $U \neq \emptyset$,

B.3.1. using Lemma 3 with $W = V$, estimate for each $a \in U$, the number p_a of vertices within distance $(1+\varepsilon)^i$ from a.

B.3.2. for each $a \in U$,

B.3.2.1. if $(\text{f-cost}(a) - f_a) < ((1+\varepsilon)^{i+1} - (1+\varepsilon)^i)p_a$,

B.3.2.1.1. set $r_a = (1+\varepsilon)^i$

B.3.2.1.2. remove a from U

else

B.3.2.1.1. $f_a = f_a + ((1+\varepsilon)^{i+1} - (1+\varepsilon)^i)p_a$

Algorithm C Implements phase 2, constructing the set S.

C.1.set $S = \emptyset$

C.2.for $i = 1, 2 \ldots$,

C.2.1. let W be the set of vertices $a \in V$ with $r_a = (1+\varepsilon)^i$

C.2.2. remove from W all vertices within distance $2(1+\varepsilon)^i$ from S

C.2.3. using Proposition 1, construct a maximal dist-$2(1+\varepsilon)^i$ independent set U from W

C.2.4. add U to S

Theorem 4. *We can solve the facility location problem within a factor* $3 + o(1)$ *from optimality in* $\tilde{O}(m)$ *time.*

Proof. We have used approximate distances in the sense of rounding up to nearest power of $(1+\varepsilon)^i$, and we used approximate facility costs in the sense that the p_a may be off by a factor $(1 \pm o(1))$. The implicit rounding down of the r_a was seen above to have no consequence. In phase 2, we made no further approximation, so with $\varepsilon = o(1)$, our total costs are only off by a factor $(1 \pm o(1))$ relative to the factor 3 in Lemma 4.

The time bound follows from the time bound in Proposition 1.

References

1. M. Charikar, S. Guha, E. Tardos, and D.B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proc. 31th STOC*, pages 1–10, 1999.

2. E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. System Sci.*, 55(3):441–453, 1997.
3. E. Cohen and U. Zwick. All-pairs small-stretch paths. In *Proc. 8th SODA*, pages 93–102, 1999.
4. M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34:596–615, 1987.
5. A. Goel, P. Indyk, and K. Varadarajan. Reductions among high demensional proximity problems. In *Proc. 10th SODA*, pages 769–778, 2001.
6. T. F. Gonzales. Clustering to minimize the maximum intercluster distance. *Theor. Comp. Sci.*, 38:293–550, 1985.
7. S. Guha, M. Mishra, R. Motwani, and L O'Callaghan. Clustering data streams. In *Proc. 41th FOCS*, pages 359–366, 2000.
8. D. Hochbaum and D. B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33:533–550, 1986.
9. W.L. Hsu and G.L. Nemhauser. Easy and hard bottleneck problems. *Discr. Appl. Math.*, 1:209–216, 1979.
10. P. Indyk. Sublinear time algorithms for metric space problems. In *Proc. 31th STOC*, pages 428–434, 1999.
11. P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the course of dimensionality. In *Proc. 30th STOC*, pages 604–613, 1998.
12. P. Indyk and M. Thorup. Approximate 1-medians, 2000.
13. K. Jain and V.V. Vazirani. Primal-dual approximation algorihtms for metric facility location and k-median problems. In *Proc. 40th FOCS*, pages 2–13, 1999. The running times involve a certain factor L that will be removed in the journal version to appear in *J. ACM*.
14. M. Luby. A simple parallel algorithm for the maiximal independent set. *SIAM J. Comput.*, 15:1036–1053, 1986.
15. R.R. Mettu and C. G. Plaxton. The online medan problem. In *Proc. 41th FOCS*, pages 339–348, 2000.
16. B.C. Tansel, R.L. Francis, and T.J. Lowe. Location on networks: A survey. part 1 and 2. *Management Science*, 29(4):482–511, 1983.
17. M. Thorup. Undirected single source shortest paths with positive integer weights in linear time. *J. ACM*, 46:362–394, 1999.
18. M. Thorup. On RAM priority queues. *SIAM J. Comput.*, 30(1):86–109, 2000.
19. M. Thorup and U. Zwick. Approximate distance oracles, 2000. Accepted for STOC'01.

Parameterized Complexity: Exponential Speed-Up for Planar Graph Problems

Jochen Alber*, Henning Fernau, and Rolf Niedermeier

Universität Tübingen, Wilhelm-Schickard-Institut für Informatik,
Sand 13, 72076 Tübingen, Fed. Rep. of Germany,
{alber,fernau,niedermr}@informatik.uni-tuebingen.de

Abstract. A parameterized problem is *fixed parameter tractable* if it admits a solving algorithm whose running time on input instance (I, k) is $f(k) \cdot |I|^{\alpha}$, where f is an arbitrary function depending only on k. Typically, f is some exponential function, e.g., $f(k) = c^k$ for constant c. We describe general techniques to obtain growth of the form $f(k) = c^{\sqrt{k}}$ for a large variety of planar graph problems. The key to this type of algorithm is what we call the "Layerwise Separation Property" of a planar graph problem. Problems having this property include PLANAR VERTEX COVER, PLANAR INDEPENDENT SET, and PLANAR DOMINATING SET.

1 Introduction

While many problems of practical interest tend to be intractable from a standard complexity-theoretic point of view, in many cases such problems have natural "structural" parameters, and practically relevant instances are often associated with "small" values of these parameters. The notion of fixed parameter tractability [10] tries to capture this intuition. This is done by taking into account solving algorithms that are exponential with respect to the parameter, but otherwise have polynomial time complexity. That is, on input instance (I, k) one terms a (parameterized) problem *fixed parameter tractable* if it allows for a solving algorithm running in time $f(k)n^{O(1)}$, where f is an arbitrary function only depending on k and $n = |I|$. The associated complexity class is called FPT. As fixed parameter tractability explicitly allows for exponential time complexity concerning the parameter, the pressing challenge is to keep the related "combinatorial explosion" as small as possible. In this paper, we provide a general framework for NP-hard planar graph problems that allows us to go from typically time $c^k n^{O(1)}$ algorithms to time $c^{\sqrt{k}} n^{O(1)}$ algorithms (subsequently briefly denoted by "$c^{\sqrt{k}}$-algorithms"), meaning an exponential speed-up.[1] The main contributions of our work, thus, are

* Supported by the Deutsche Forschungsgemeinschaft (research project PEAL (Parameterized complexity and Exact ALgorithms), NI 369/1-1).

[1] Actually, whenever we can construct a so-called problem kernel of polynomial size in polynomial time (which is often the case for parameterized problems), then we can replace the term $c^{\sqrt{k}} n^{O(1)}$ by $c^{\sqrt{k}} k^{O(1)} + n^{O(1)}$.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 261–272, 2001.

- to provide new results and a "structural breakthrough" for the parameterized complexity of a large class of problems,
- to parallel and complement results for the approximability of planar graph problems obtained by Baker [4],
- to methodize and extend previous work on concrete graph problems [1], and
- to systematically compute the bases in the exponential terms.

Fixed parameter tractability. The parameterized tractability approach tries to restrict the seemingly inherent "combinatorial explosion" of NP-hard problems to a "small part" of the input, the parameter. For instance, VERTEX COVER allows for an algorithm with running time $O(kn + 1.3^k)$, where k is the size of the vertex cover to be constructed [8,14]. One direction in current research is to investigate problems with fixed parameter algorithms of running time $c^k n^{O(1)}$ and to try to get the constant c as small as possible. Getting small constant bases in the exponential factor $f(k)$ is also our concern, but here, we focus on functions f (asymptotically) growing as slowly as possible.

Planar graph problems. Planar graphs build a natural and practically important graph class. Many problems that are NP-complete for general graphs (such as VERTEX COVER and DOMINATING SET) remain so when restricted to planar graphs. Whereas many NP-complete graph problems are hard to approximate in general graphs, Baker, in her well-known work [4], showed that many of them possess a polynomial time approximation scheme for planar graphs. However, the degree of the polynomial grows with the quality of the approximation. Alternatively, finding an "efficient" exact solution in "reasonable exponential time" is an interesting and promising research challenge.

Relations to previous work. In recent work, algorithms were presented that constructively produce a solution for PLANAR DOMINATING SET and related problems in time $c^{\sqrt{k}}n$ [1]. To obtain these results, it was proven that the treewidth of a planar graph with a dominating set of size k is bounded by $O(\sqrt{k})$, and that a corresponding tree decomposition can be found in time $O(\sqrt{k}n)$. Building on that problem-specific work with its rather tailor-made approach for dominating sets, here, we take a much broader perspective. From a practitioner's point of view, this means that, since the algorithms developed here can be stated in a very general framework, only small parts have to be changed to adapt them to the concrete problem. In this sense, our work differs strongly from research directions where running times of algorithms are improved in a very problem-specific manner (e.g., by extremely sophisticated case-distinctions, as in the case of VERTEX COVER for general graphs). For example, once one can show that a problem has the so-called "Layerwise Separation Property," one can run a general algorithm which quickly computes a tree decomposition of guaranteed small width (independent of the concrete problem).

Results. We provide a general methodology for the design of $c^{\sqrt{k}}$-algorithms. A key to this is the notion of select&verify graph problems and the introduction of the Layerwise Separation Property (see Section 3) of such problems in connection with the concept of linear problem kernels (see Subsection 2.1). We show that problems that have the Layerwise Separation Property and admit ei-

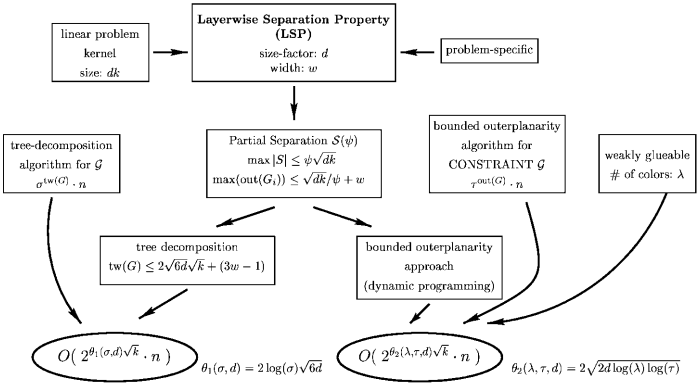


Fig. 1. Roadmap of our methodology for planar graph problems.

ther a tree decomposition based algorithm (cf., e.g., [17]) or admit an algorithm based on bounded outerplanarity (cf. [4]), can be solved in time $c^{\sqrt{k}}n^{O(1)}$. For instance, these include PLANAR VERTEX COVER, PLANAR INDEPENDENT SET, PLANAR DOMINATING SET, or PLANAR EDGE DOMINATION and also variations of these, such as their weighted versions. Moreover, we give explicit formulas to determine the base c of the exponential term with respect to the problem specific parameters. For PLANAR VERTEX COVER, e.g., we obtain a time $O(2^{4\sqrt{3k}}n)$ algorithm. The methods can be generalized in a way that basically all FPT-problems that admit tree-decomposition based algorithms can be attacked with our approach. A library containing implementations of various algorithms sketched in this paper is currently under development. It uses the LEDA package [13] for graph algorithms and the results obtained so far are encouraging.

Review of presented methodology. In a first phase, one separates the graph in a particular way ("layerwise"). The key property of a graph problem which allows such an approach will be the so-called "Layerwise Separation Property." Corresponding details are presented in Section 3. It will be shown that such a property holds for quite a large class of graph problems. In a second phase, the problem is solved on the layerwisely separated graph. We present two independent ways to achieve this in Section 4; either using the separators to set up a tree decomposition of width $O(\sqrt{k})$ and solving the problem using this tree decomposition, or using a combination of a trivial approach on the separators and some algorithms working on graphs of bounded outerplanarity (see [4]) for the partitioned rest graphs. Figure 1 gives a general overview of our methodology.

Several details and proofs had to be deferred to the full version [2].

2 Basic Definitions and Preliminaries

We consider undirected graphs $G = (V, E)$, V denoting the vertex set and E denoting the edge set. Sometimes we refer to V by $V(G)$. Let $G[D]$ denote

the subgraph induced by a vertex set D. We only consider simple (no double edges) graphs without self-loops. We study *planar* graphs, i.e., graphs that can be drawn in the plane without edge crossings. Let (G, ϕ) denote a *plane* graph, i.e., a planar graph G together with an embedding ϕ. A *face* of a plane graph is any topologically connected region surrounded by edges of the plane graph. The one unbounded face of a plane graph is called the *exterior face*. We study the following "graph numbers": A *vertex cover* C of a graph G is a set of vertices such that every edge of G has at least one endpoint in C; the size of a vertex cover set with a minimum number of vertices is denoted by $vc(G)$. An *independent set* of a graph G is a set of pairwise nonadjacent vertices; the size of an independent set with a maximum number of vertices is denoted by $is(G)$. A *dominating set* D of a graph G is a set of vertices such that each of the vertices in G lies in D or has at least one neighbor in D; the size of a dominating set with a minimum number of vertices is denoted by $ds(G)$. The corresponding problems are (PLANAR) VERTEX COVER, INDEPENDENT SET, and DOMINATING SET.

2.1 Linear Problem Kernels

Reduction to problem kernel is a core technique for the development of fixed parameter algorithms (see [10]). In a sense, the idea behind is to cut off the "easy parts" of a given problem instance such that only the "hard kernel" of the problem remains, where, then, e.g., exhaustive search can be applied (with reduced costs).

Definition 1. *Let $\mathcal{L}$ be a parameterized problem, i.e., $\mathcal{L}$ consists of pairs (I, k), where problem instance I has a solution of size k (the parameter).*[2] Reduction to problem kernel[3] *then means to replace problem (I, k) by a "reduced" problem (I', k') (which we call the* problem kernel*) such that $k' \leq c \cdot k$ and $|I'| \leq q(k')$ with constant c, polynomial q, and $(I, k) \in \mathcal{L}$ iff $(I', k') \in \mathcal{L}$. Furthermore, we require that the reduction from (I, k) to (I', k') (that we call* kernelization*) is computable in polynomial time $T_K(|I|, k)$.*

Usually, having constructed a size $k^{O(1)}$ problem kernel in time $n^{O(1)}$, one can improve the time complexity $f(k)n^{O(1)}$ of a fixed parameter algorithm to $f(k)k^{O(1)} + n^{O(1)}$. Subsequently, our focus is on decreasing $f(k)$, and we do not always refer to this simple fact. Often (cf. the subsequent example VERTEX COVER), the best one can hope for the problem kernel is size linear in k, a so-called *linear problem kernel*. For instance, using a theorem of Nemhauser and Trotter [15], Chen *et al.* [8] recently observed a problem kernel of size $2k$ for VERTEX COVER on general (not necessarily planar) graphs. According to the current state of knowledge, this is the best one could hope for. As a further example, note that due to the four color theorem for planar graphs and the

[2] In this paper, we assume the parameter to be a positive integer, although, in general, it might also be from an arbitrary language (e.g., being a subgraph).

[3] Here, we give a somewhat "restricted definition" of reduction to problem kernel which, however, applies to all practical cases we know.

corresponding algorithm generating a four coloring [16], it is easy to see that PLANAR INDEPENDENT SET has a problem kernel of size $4k$.

Besides the positive effect of reducing the input size significantly, this paper gives further justification, in particular, for the importance of linear problem kernels. The point is that once we have a linear problem kernel, e.g., for PLANAR VERTEX COVER or PLANAR INDEPENDENT SET, it is fairly easy to get $c^{\sqrt{k}}$-algorithms for these problems based upon the famous planar separator theorem [12]. The constant factor in the problem kernel size directly influences the value of the exponential base and hence, lowering the kernel size means improved efficiency. We will show alternative, more efficient ways (without using the planar separator theorem) of how to make use of linear problem kernels in a generic way in order to obtain $c^{\sqrt{k}}$-algorithms for planar graph problems.

2.2 Tree Decomposition and Layer Decomposition of Graphs

Definition 2. *A* tree decomposition *of a graph $G = (V, E)$ is a pair $\langle \{X_i \mid i \in I\}, T\rangle$, where $X_i \subseteq V$ is called a* bag *and T is a tree with the elements of I as nodes, such that the following hold:*

1. $\bigcup_{i \in I} X_i = V$;
2. *for every edge $\{u, v\} \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$;*
3. *for all $i, j, k \in I$, if j lies on the path between i and k in T, then $X_i \cap X_k \subseteq X_j$.*

The width *of $\langle \{X_i \mid i \in I\}, T\rangle$ is $\max\{|X_i| \mid i \in I\} - 1$. The* treewidth $\mathrm{tw}(G)$ *of G is the minimum ℓ such that G has a tree decomposition of width ℓ.*

Details on tree decompositions can be found in [5,6,11]. Let $G = (V, E)$ be a planar graph. The vertices of G can be decomposed according to the level of the "layer" in which they appear in an embedding ϕ, see [1,4].

Definition 3. *Let $(G = (V, E), \phi)$ be a plane graph.*

a) The layer decomposition *of (G, ϕ) is a disjoint partition of the vertex set V into sets $L_1, \ldots, L_r$, which are recursively defined as follows:*
- *L_1 is the set of vertices on the exterior face of G, and*
- *L_i is the set of vertices on the exterior face of $G[V - \bigcup_{j=1}^{i-1} L_j]$ for $i = 2, \ldots r$.*

We will denote the layer decomposition of (G, ϕ) by $\mathcal{L}(G, \phi) := (L_1, \ldots, L_r)$.

b) The set L_i is called the ith layer *of (G, ϕ).*

c) The (uniquely defined) number r of different layers is called the outerplanarity *of (G, ϕ), denoted by $\mathrm{out}(G, \phi) := r$.*

d) We define $\mathrm{out}(G)$ to be the smallest outerplanarity possible among all plane embeddings, i.e., minimizing over all plane embeddings ϕ of G we set

$$\mathrm{out}(G) := \min_{\phi} \mathrm{out}(G, \phi).$$

Proposition 1 ([1]). *Let $(G = (V, E), \phi)$ be a plane graph. The layer decomposition $\mathcal{L}(G, \phi) = (L_1, \ldots, L_r)$ can be computed in time $O(|V|)$.*

2.3 Algorithms Based on Separators in Graphs

One of the most useful algorithmic techniques for solving computational problems is divide-and-conquer. To apply this technique to planar graphs, we need graph separators and related notions.

Graph separators and select&verify problems. Graph separators are defined as follows. Let $G = (V, E)$ be an undirected graph. A *separator* $S \subseteq V$ of G partitions V into two sets A and B such that $A \cup B \cup S = V$ with $A \cap B = A \cap S = B \cap S = \emptyset$ and no edge joins vertices in A and B. In general, of course, A, B and S will be non-empty.

Definition 4. *A set $\mathcal{G}$ of tuples (G, k), G an undirected graph with vertex set $V = \{v_1, \ldots, v_n\}$ and k a positive real number, is called a* select&verify graph problem *if there exists a pair $(P., \mathrm{opt})$ with $\mathrm{opt} \in \{\min, \max\}$, such that $P.$ is a function that assigns to an undirected graph G (with n vertices) a polynomial time computable function $P_G : \{0,1\}^n \to \mathbb{R}_+ \cup \{\pm\infty\}$, such that*

$$(G,k) \in \mathcal{G} \quad \Leftrightarrow \quad \begin{cases} \mathrm{opt}_{\boldsymbol{x}\in\{0,1\}^n} P_G(\boldsymbol{x}) \le k & \text{if } \mathrm{opt} = \min \\ \mathrm{opt}_{\boldsymbol{x}\in\{0,1\}^n} P_G(\boldsymbol{x}) \ge k & \text{if } \mathrm{opt} = \max . \end{cases}$$

It is an easy observation that every select&verify graph problem that additionally admits a linear problem kernel of size dk is solvable in time $O(2^{dk}k + T_K(n,k))$.

Vertex cover is an easy example for a select&verify graph problem. Here, for $G = (V, E)$, one may use (with the convention $\infty \cdot 0 = 0$)

$$P_G(\boldsymbol{x}) = \sum_{i=1}^{n} x_i + \sum_{\{v_i,v_j\}\in E} \infty \cdot (1 - x_i)(1 - x_j).$$

Algorithms based on separator theorems. Lipton and Tarjan [12] have used their famous separator theorem in order to design algorithms with a running time of $O(c^{\sqrt{n}})$ for certain select&verify planar graph problems. This naturally implies that, in the case of parameterized planar graph problems for which a linear kernel is known, algorithms with running time $O(c'^{\sqrt{k}} + T_K(n,k))$ can be derived. As worked out in [3], a straightforward application yields very bad constants, even when dealing with improved versions of the planar separator theorem (see [9]); for instance, $c' = 2^{15.1823} \approx 40000$ for planar vertex cover. We will see algorithms with much better constants in this paper. In addition, the advantages of the approach pursued in this paper also lie in weaker assumptions. In some cases, we may drop requirements such as linear problem kernels by replacing it with the so-called "Layerwise Separation Property," a seemingly less restrictive demand.

3 Phase 1: Layerwise Separation

We will exploit the layer-structure of a plane graph in order to gain a "nice" separation of the graph. It is important that a "yes"-instance (G, k) (where G is

a plane graph) of the graph problem $\mathcal{G}$ admits a so-called "layerwise separation" of small size. By this, we mean, roughly speaking, a separation of the plane graph G (i.e., a collection of separators for G), such that each separator is contained in the union of constantly many subsequent layers (see conditions 1 and 2 of the following definition). For (fixed parameter) algorithmic purposes, it will be important that the corresponding separators are "small" (see condition 3 below).

Definition 5. *Let $(G = (V, E), \phi)$ be a plane graph of outerplanarity $r := \mathrm{out}(G, \phi)$ and let $\mathcal{L}(G, \phi) = (L_1, \ldots, L_r)$ be its layer decomposition. A* layerwise separation of width w and size s *of (G, ϕ) is a sequence $(S_1, \ldots, S_r)$ of subsets of V, with the properties that:*
1. $S_i \subseteq \bigcup_{j=i}^{i+(w-1)} L_j$, 2. S_i separates layers L_{i-1} and L_{i+w}, and 3. $\sum_{j=1}^{r} |S_j| \leq s$.

Definition 6. *A parameterized problem $\mathcal{G}$ for planar graphs is said to have the* Layerwise Separation Property *(abbreviated by: LSP) of width w and size-factor d if for each $(G, k) \in \mathcal{G}$ and every planar embedding ϕ of G, the plane graph (G, ϕ) admits a layerwise separation of width w and size dk.*

3.1 How Can Layerwise Separations Be Obtained?

The Layerwise Separation Property can be shown directly for many parameterized graph problems. As an example, consider PLANAR VERTEX COVER. Here, we get constants $w = 2$ and $d = 2$. In fact, for $(G, k) \in$ VERTEX COVER (where (G, ϕ) is a plane graph) with a "witnessing" vertex cover V' of size k, the sets $S_i := (L_i \cup L_{i+1}) \cap V'$ form a layerwise separation, given the layer decomposition $\mathcal{L}(G, \phi) = (L_1, \ldots, L_r)$. In [1], the non-trivial fact is proven that for PLANAR DOMINATING SET, the LSP holds with constants $w = 3$ and $d = 51$.

Lemma 1. *Let $\mathcal{G}$ be a parameterized problem for planar graphs that admits a problem kernel of size dk. Then, the parameterized problem $\mathcal{G}'$ where each instance is replaced by its problem kernel has the LSP of width 1 and size-factor d.*

With Lemma 1 and the size $2k$ problem kernel for VERTEX COVER (see Subsection 2.1), we derive, for example, that PLANAR VERTEX COVER has the LSP of width 1 and size-factor 2 (which is even better than what was shown above). Using the $4k$ problem kernel for PLANAR INDEPENDENT SET, we see that this problem has the LSP of width 1 and size-factor 4 on the set of reduced instances.

3.2 What Are Layerwise Separations Good for?

The idea of the following is that, from a layerwise separation of small size (say bounded by $O(k)$), we are able to choose a set of separators such that their size is bounded by $O(\sqrt{k})$ and—at the same time—the subgraphs into which these separators cut the original graph have outerplanarity bounded by $O(\sqrt{k})$.

Definition 7. *Let $(G = (V, E), \phi)$ be a plane graph with layer decomposition $\mathcal{L}(G, \phi) = (L_1, \ldots, L_r)$. A* partial layerwise separation of width w *is a sequence $\mathcal{S} = (S_1, \ldots, S_q)$ such that there exist $i_0 = 1 \leq i_1 < \ldots < i_q \leq r = i_{q+1}$ such that for $i = 1, \ldots, q$:*[4]

[4] By default, we set $S_i := \emptyset$ for $i < 1$ and $i > q$.

1. $S_j \subseteq \bigcup_{\ell=i_j}^{i_j+(w-1)} L_\ell$,
2. $i_j + w \leq i_{j+1}$ *(so the sets in* $\mathcal{S}$ *are pairwise disjoint), and*
3. S_j *separates layers* L_{i_j-1} *and* L_{i_j+w}.

The sequence $\mathcal{C}_{\mathcal{S}} = (G_0, \ldots, G_q)$ *with*

$$G_j := G[(\bigcup_{\ell=i_j}^{i_{j+1}+(w-1)} L_\ell) - (S_j \cup S_{j+1})], \quad j = 0, \ldots, q$$

is called the sequence of graph chunks *obtained by* $\mathcal{S}$.

Theorem 1. *Let* $(G = (V, E), \phi)$ *be a plane graph that admits a layerwise separation of width* w *and size* dk. *Then, for every* $\psi \in \mathbb{R}_+$, *there exists a partial layerwise separation* $\mathcal{S}(\psi)$ *of width* w *such that*

1. $\max_{S \in \mathcal{S}(\psi)} |S| \leq \psi\sqrt{dk}$ *and*
2. $\mathrm{out}(H) \leq \frac{\sqrt{dk}}{\psi} + w$ *for each graph chunk* H *in* $\mathcal{C}_{\mathcal{S}(\psi)}$.

Moreover, there is an algorithm with running time $O(\sqrt{k}n)$ *which, for given* ψ, *recognizes whether* (G, ϕ) *admits a layerwise separation of width* w *and size* dk *and, if so, computes* $\mathcal{S}(\psi)$.

Proof. (Sketch) For $m = 1, \ldots, w$, consider the integer sequences $I_m = (m + jw)_{j=0}^{\lfloor r/w \rfloor - 1}$ and the corresponding sequences of separators $\mathcal{S}_m = (S_i)_{i \in I_m}$. Note that each $\mathcal{S}_m$ is a sequence of pairwise disjoint separators. Since $(S_1, \ldots, S_r)$ is a layerwise separation of size dk, this implies that there exists a $1 \leq m' \leq w$ with $\sum_{i \in I_{m'}} |S_i| \leq \frac{dk}{w}$ $(*)$.

For a given ψ, let $s := \psi\sqrt{dk}$. Define $\mathcal{S}(\psi)$ to be the subsequence of $\mathcal{S}_{m'}$ such that $|S| \leq s$ for all $S \in \mathcal{S}(\psi)$, and $|S| > s$ for all $S \in \mathcal{S}_{m'} - \mathcal{S}(\psi)$. This yields condition 1. As to condition 2, suppose that $\mathcal{S}(\psi) = (S_{i_1}, \ldots, S_{i_q})$. The number of separators in $\mathcal{S}_{m'}$ that appear between S_{i_j} and $S_{i_{j+1}}$ is $(i_{j+1} - i_j)/w$. Since all of these separators have size $\geq s$, their number has to be bounded by dk/ws, see $(*)$. Therefore, $i_{j+1} - i_j \leq \sqrt{dk}/\psi$ for all $j = 1, \ldots, q-1$. Hence, the chunks $G[(\bigcup_{\ell=i_j}^{i_{j+1}+(w-1)} L_\ell) - (S_{i_j} \cup S_{i_{j+1}})]$ have outerplanarity at most $\sqrt{dk}/\psi + w$.

The proof can be turned into a constructive algorithm. This is outlined in the full version [2]. □

4 Phase 2: Algorithms on Layerwisely Separated Graphs

After Phase 1, we are left with a set of disjoint (layerwise) separators of size $O(\sqrt{k})$ separating the graph in components, each of which having outerplanarity bounded by $O(\sqrt{k})$.

4.1 Using Tree Decompositions

We will show how the existence of a layerwise separation of small size helps to constructively obtain a tree decomposition of small width. The following result can be found in [6, Theorem 83] and [1, Theorem 12].

Proposition 2. *For a plane graph (G, ϕ), we have* $\mathrm{tw}(G) \leq 3 \cdot \mathrm{out}(G) - 1$. *Such a tree decomposition can be found in* $O(\mathrm{out}(G) \cdot n)$ *time.*

Theorem 2. *Let (G, ϕ) be a plane graph that admits a layerwise separation of width w and size dk. Then, we have* $\mathrm{tw}(G) \leq 2\sqrt{6dk} + (3w - 1)$. *Such a tree decomposition can be computed in time* $O(k^{3/2}n)$.

Proof. (Sketch) By Theorem 1, for each $\psi \in \mathbb{R}_+$, there exists a partial layerwise separation $\mathcal{S}(\psi) = (S_1, \ldots, S_q)$ of width w with corresponding graph chunks $\mathcal{C}_{\mathcal{S}(\psi)} = (G_0, \ldots, G_q)$, such that $\max_{S \in \mathcal{S}(\psi)} |S| \leq \psi\sqrt{dk}$ and $\mathrm{out}(G_i) \leq \sqrt{dk}/\psi + w$ for $i = 0, \ldots, q$. The algorithm that constructs a tree decomposition $\mathfrak{X}_\psi$ is:

1. Construct a tree decomposition $\mathcal{X}_i$ of width at most $3\,\mathrm{out}(G_i) - 1$ for each of the graphs G_i (using the algorithm from Proposition 2).
2. Add S_i and S_{i+1} to every bag in $\mathcal{X}_i$ $(i = 0, \ldots, q)$.
3. Let T_i be the tree of $\mathcal{X}_i$. Then, successively add an arbitrary connection between the trees T_i and T_{i+1} in order to obtain a tree T.

The tree T, together with the constructed bags, gives a tree decomposition of G, see [1, Prop. 4]. Its width $\mathrm{tw}(\mathfrak{X}_\psi)$ is upperbounded by $(2\psi + 3/\psi)\sqrt{dk} + (3w - 1)$, which is minimal if $\psi = \sqrt{3/2}$. Therefore, $\mathrm{tw}(\mathfrak{X}_\psi) \leq 2\sqrt{6dk} + (3w - 1)$. □

For example, Theorem 2 and previous observations imply $\mathrm{tw}(G) \leq 4\sqrt{3\,vc(G)} + 5$ and $\mathrm{tw}(G) \leq 6\sqrt{34\,ds(G)} + 8$ for planar graphs G. Note that for general graphs, no relation of the form $\mathrm{tw}(G) \leq f(ds(G))$ (for any function f) holds. For VERTEX COVER, only the linear relation $\mathrm{tw}(G) \leq vc(G)$ can be shown easily.

In addition, Theorem 2 yields a $c^{\sqrt{k}}$-algorithm for certain graph problems.

Theorem 3. *Let $\mathcal{G}$ be a parameterized problem for planar graphs. Suppose that $\mathcal{G}$ has the LSP of width w and size-factor d and that there exists a time $\sigma^\ell n$ algorithm that decides $(G, k) \in \mathcal{G}$, if G is given together with a tree decomposition of width ℓ.*

Then, there is an algorithm to decide $(G, k) \in \mathcal{G}$ in time $O(\sigma^{3w-1} \cdot 2^{\theta_1(\sigma,d)\sqrt{k}}n)$, *where* $\theta_1(\sigma, d) = 2\log(\sigma)\sqrt{6d}$.

Proof. In time $O(\sqrt{k}n)$ (see Theorem 1), we can check whether an instance (G, k) admits a layerwise separation of width w and size dk. If so, the algorithm of Theorem 2 computes a tree decomposition of width at most $2\sqrt{6dk} + (3w - 1)$, and we can decide $(G, k) \in \mathcal{G}$ by using the given tree decomposition algorithm in time $O(\sigma^{2\sqrt{6dk}+(3w-1)}n)$. If (G, k) does not admit such a layerwise separation, we know that $(G, k) \notin \mathcal{G}$, by definition of LSP. □

Going back to our running examples, it is well-known that PLANAR VERTEX COVER and PLANAR INDEPENDENT SET admit such a tree decomposition based algorithm for $\sigma = 2$. For PLANAR VERTEX COVER, we have seen that the LSP of width 1 and size-factor $d = 2$ holds. Hence, Theorem 3 guarantees an $O(2^{4\sqrt{3k}}n)$ algorithm for this problem. For PLANAR INDEPENDENT SET, we have a linear problem kernel of size $4k$, hence, the LSP of width 1 and size-factor $d = 4$ holds, which yields an $O(2^{4\sqrt{6k}}n)$ algorithm.

4.2 Using Bounded Outerplanarity

We now turn our attention to select&verify problems subject to the assumption that a solving algorithm of linear running time on the class of graphs of bounded outerplanarity exists. This issue was addressed in [4]; a variety of examples can be found therein. We examine how, in this context, the notions of select&verify problems and the LSP will lead to $c^{\sqrt{k}}$-algorithms.

Due to the lack of space, we only give an intuitive explanation of the notions "weak glueability" and "CONSTRAINT $\mathcal{G}$" associated to a select&verify problem $\mathcal{G}$ which appear in the formulation of the following results. For a more detailed definition we refer to the long version [2] or to [3]. A problem $\mathcal{G}$ is *weakly glueable with λ colors* if a solution of $\mathcal{G}$ on an instance G can be obtained by "merging" solutions of CONSTRAINT $\mathcal{G}$ with $G[A \cup S]$ and $G[B \cup S]$, where S separates G into two parts A and B. Here, CONSTRAINT $\mathcal{G}$ is a variant of $\mathcal{G}$, in which it is already fixed which vertices of S belong to an admissible solution. The number λ, in some sense, measures the complexity of the merging step. For example, PLANAR VERTEX COVER, and PLANAR INDEPENDENT SET are weakly glueable with $\lambda = 2$ colors and, PLANAR DOMINATING SET is weakly glueable with "essentially" $\lambda = 3$ colors.

Similar to Theorem 3, we construct a partial layerwise separation $\mathcal{S}(\psi)$ with optimally adapted trade-off parameter ψ to enable an efficient dynamic programming algorithm. We omit the proof of the following theorem (see [2] for details).

Theorem 4. *Let $\mathcal{G}$ be a select&verify problem for planar graphs. Suppose that $\mathcal{G}$ has the LSP of width w and size-factor d, that $\mathcal{G}$ is weakly glueable with λ colors, and that there exists an algorithm that solves the problem* CONSTRAINT $\mathcal{G}$ *for a given graph G in time $\tau^{\mathrm{out}(G)}n$.*

Then, there is an algorithm to decide $(G, k) \in \mathcal{G}$ in time $O(\tau^w \cdot 2^{\theta_2(\lambda,\tau,d)\sqrt{k}}n)$, where $\theta_2(\lambda, \tau, d) = 2\sqrt{2d\log(\lambda)\log(\tau)}$.

It remains to say for which problems there exists a solving algorithm of the problem CONSTRAINT $\mathcal{G}$ for a given graph G in time $\tau^{\mathrm{out}(G)}n$. For PLANAR VERTEX COVER, we have $d = 2$, $w = 1$ and $\tau = 8$ (see the result of Baker [4] which can be adapted to the constraint case fairly easily) and, hence, the approach in Theorem 4 yields an $O(2^{4\sqrt{3k}}n)$ time algorithm.

As an alternative to Baker, we again may use tree decomposition based approaches: Let $\mathcal{G}$ be a parameterized problem for planar graphs. Suppose that

there exists a time $\sigma^{\ell}n$ algorithm that solves CONSTRAINT $\mathcal{G}$, when G is given together with a tree decomposition of width ℓ. Then, due to Proposition 2, there is an algorithm that solves CONSTRAINT $\mathcal{G}$ in time $\tau^{\text{out}(G)}n$ for $\tau = \sigma^3$.

The following easy corollary helps comparing the approach from Subsection 4.1 (i.e., Theorem 3) with the approach in this subsection (i.e., Theorem 4).

Corollary 1. *Let $\mathcal{G}$ be a select&verify problem for planar graphs. Suppose that $\mathcal{G}$ has the LSP of width w and size-factor d, that $\mathcal{G}$ is weakly glueable with λ colors, and that there exists a time $\sigma^{\ell}n$ algorithm that solves* CONSTRAINT $\mathcal{G}$ *for a graph G, if G is given together with a tree decomposition of width ℓ.*

Then, there is an algorithm to decide $(G,k) \in \mathcal{G}$ in time $O(\sigma^{3w} \cdot 2^{\theta_3(\lambda,\sigma,d)\sqrt{k}}n)$, where $\theta_3(\lambda,\sigma,d) = 2\sqrt{6d\log(\lambda)\log(\sigma)}$.

The exponential factor of the algorithm in Corollary 1, i.e., $\theta_3(\lambda,\sigma,d)$, is related to the corresponding exponent of Theorem 3, i.e., $\theta_1(\sigma,d)$, in the following way: $\sqrt{\log\lambda} \cdot \theta_1(\sigma,d) = \sqrt{\log\sigma} \cdot \theta_3(\lambda,\sigma,d)$. From this, we derive that, if $\lambda > \sigma$, the algorithm in Theorem 3 outperforms the one of Corollary 1, whereas, if $\lambda < \sigma$, the situation is vice versa. However, in order to apply Corollary 1, we need the three extra assumptions that we have a select&verify problem which is weakly glueable and that we can deal with the problem CONSTRAINT $\mathcal{G}$ in the treewidth algorithm.

5 Conclusion

To some extent, this paper can be seen as the "parameterized complexity counterpart" to what was developed by Baker [4] in the context of approximation algorithms. We describe two main ways (namely linear problem kernels and problem-specific approaches) to achieve the novel concept of Layerwise Separation Property, from which again, two approaches (tree decomposition and bounded outerplanarity) lead to $c^{\sqrt{k}}$-algorithms for planar graph problems (see Figure 1 for an overview). A slight modification of our presented techniques can be used to extend our results to parameterized problems that admit a problem kernel of size $p(k)$ (not necessarily linear!). In this case, the running time can be sped up from $2^{O(p(k))}n^{O(1)}$ to $2^{O(\sqrt{p(k)})}n^{O(1)}$ (see [2] for details). Basically all FPT-problems that admit treewidth based algorithms can be handled by our methods (see [17]).

Future research topics raised by our work include to further improve the ("exponential") constants, e.g., by a further refined and more sophisticated "layer decomposition tree"; to investigate and extend the availability of linear problem kernels for all kinds of planar graph problems; to provide implementations of our approach accompanied by sound experimental studies, thus taking into account that all our analysis is worst case and often overly pessimistic. Finally, a more general question is whether there are other "problem classes" that allow for $c^{\sqrt{k}}$ fixed parameter algorithms. Cai and Juedes [7], however, very recently showed the surprising result that for a list of parameterized problems (e.g., VERTEX COVER on general graphs) $c^{o(k)}$-algorithms are impossible unless FPT $= W[1]$.

Acknowledgements. We'd like to mention that parts of this work were discussed at the first international Workshop on Parameterized Complexity (organized by Mike Fellows and Venkatesh Raman) in Chennai, India, December 7–9, 2000.

References

1. J. Alber, H. Bodlaender, H. Fernau, and R. Niedermeier. Fixed parameter algorithms for planar dominating set and related problems. In *Proc. 7th SWAT*, vol. 1851 of *LNCS*, Springer, pp. 97–110, 2000. Full version available as Technical Report UU-CS-2000-28, Utrecht University, 2000.
2. J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speed-up for planar graph problems. Technical Report TR01–023, ECCC Reports, Trier, March 2001. Available through `http://www.eccc.uni-trier.de/eccc/`.
3. J. Alber, H. Fernau, and R. Niedermeier. Graph separators: a parameterized view. To appear in *Proc. 7th COCOON*, 2001. Full version available as Technical Report WSI–2001–8, Universität Tübingen (Germany), Wilhelm-Schickard-Institut für Informatik, March 2001.
4. B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.
5. H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proc. 22nd MFCS*, vol. 1295 of *LNCS*, Springer, pp. 19–36, 1997.
6. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comp. Sci.*, 209:1–45, 1998.
7. L. Cai and D. Juedes. Subexponential parameterized algorithms collapse the W-hierarchy. In *Proc. 28th ICALP*, 2001.
8. J. Chen, I. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. In *Proc. 25th WG*, vol. 1665 of *LNCS*, Springer, pp. 313–324, 1999.
9. H. N. Djidjev and S. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, 34:231–243, 1997.
10. R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.
11. T. Kloks. *Treewidth: Computations and Approximations*, vol. 842 of *LNCS*, Springer, 1994.
12. R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comp.*, 9(3):615–627, 1980.
13. K. Mehlhorn and S. Näher. *LEDA: A Platform of Combinatorial and Geometric Computing.* Cambridge University Press, Cambridge, England, 1999.
14. R. Niedermeier and P. Rossmanith. Upper Bounds for Vertex Cover further improved. In *Proc. 16th STACS*, vol. 1563 of *LNCS*, Springer, pp. 561–570, 1999.
15. G. L. Nemhauser and J. L. E. Trotter. Vertex packing: structural properties and algorithms. *Math. Progr.*, 8:232–248, 1975.
16. N. Robertson, D. P. Sanders, P. Seymour, and R. Thomas. Efficiently four-coloring planar graphs. In *Proc. 28th STOC*, ACM Press, pp. 571–575, 1996.
17. J. A. Telle and A. Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. In *Proc. 3rd WADS*, vol. 709 of *LNCS*, Springer, pp. 610–621, 1993.

Subexponential Parameterized Algorithms Collapse the W-Hierarchy*

Liming Cai[1] and David Juedes[2]

[1] School of Electrical Engineering and Computer Science,
Ohio University, Athens, OH 45701
cai@cs.ohiou.edu.
[2] School of Electrical Engineering and Computer Science,
Ohio University, Athens, OH 45701
juedes@ohiou.edu

Abstract. It is shown that for essentially all MAX SNP-hard optimization problems finding exact solutions in subexponential time is not possible unless $W[1] = FPT$. In particular, we show that $O(2^{o(k)}p(n))$ parameterized algorithms do not exist for Vertex Cover, Max Cut, Max c-Sat, and a number of problems on bounded degree graphs such as Dominating Set and Independent Set, unless $W[1] = FPT$. Our results are derived via an approach that uses an extended parameterization of optimization problems and associated techniques to relate the parameterized complexity of problems in FPT to the parameterized complexity of extended versions that are $W[1]$-hard.

1 Introduction

Recent substantial progress has been made in building better and better parameterized algorithms for a variety of NP-complete problems. Consider the problem of determining whether a graph with n nodes has a Vertex Cover of size k. Starting with the early work of Buss [7] who discovered a $O(2^k k^{2k+2} + kn)$ algorithm for the problem, the running time of parameterized algorithms for Vertex Cover has been improved to $O(2^k k^2 + kn)$ by Downey and Fellows [11], $O(1.325^k k^2 + kn)$ by Balasubramanian *et al.* [5], $O(1.3196^k k^2 + kn)$ by Downey, Fellows, and Stege [13], $O(1.29175^k k^2 + kn)$ by Niedermeier and Rossmanith [17], and $O(1.271^k k^2 + kn)$ by Chen, Kanj, and Jia [9]. Similar improvements have been made for other NP-complete problems [2]. In particular, we mention the case for Planar Dominating Set. As shown by Downey and Fellows [11], this problem is known to be fixed parameter tractable via a $O(11^k|G|)$ algorithm. However, this result was recently improved to $O(2^{O(\sqrt{k})}n)$ by Alber, Bodlaender, Fernau, and Niedermeier [1].

Noting the progress on algorithms for Planar Dominating Set, it is natural to ask if similar progress can be made for Vertex Cover. In particular, it

* This work was supported in part by the National Science Foundation research grant CCR-000248.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 273–284, 2001.

is natural to ask if the current $O(2^{\epsilon k}p(n))$ upper bound on VERTEX COVER can be improved to $O(2^{o(k)}p(n))$. As our main result, we show that this is unlikely since the existence of such an algorithm implies that the W-hierarchy collapses at the first level, i.e., $W[1] = FPT$. With this in mind, it is natural to ask why is it possible to build a parameterized algorithm for PLANAR DOMINATING SET that runs in time $O(2^{o(k)}p(n))$ when the existence of such an algorithm for VERTEX COVER implies that $W[1] = FPT$. The answer to this question seems to lie in the approximability of these two problems. While both of these problems are NP-complete [14], it is known that PLANAR DOMINATING SET has a PTAS [4]. The same is not true for VERTEX COVER unless $P = NP$ because VERTEX COVER is MAX SNP-hard [18]. As we show here, the fact the VERTEX COVER is MAX SNP-hard means that it does not have a subexponential parameterized algorithm unless $W[1] = FPT$.

Our results are obtained using new parameterized proof techniques. In particular, we examine generalized parameterizations of optimization problems and relate the complexities of various parameterizations. For each maximization problem Π, we define $\Pi^{(r,s)}$ to be the parameterized problem that determines whether $OPT_\Pi(I) \geq r(I) + ks(|I|)$, for functions r and s. Analogous parameterizations are defined for minimization problems. As we show here, the parameterized complexity of these problems depends largely on the function s. We show that for certain optimization problems Π such as MAX C-SAT, $\Pi^{(r,s)}$ is parameterized tractable when $s = 1$ or even $o(\log n)$, but $\Pi^{(r,s)}$ becomes $W[1]$-hard when $s = \Theta(\log n)$.

This extended abstract is structured as follows. In section 2, we provide necessary notation concerning parameterized complexity theory, and we introduce a general framework for examining parameterized versions of optimization problems. In section 2, we begin to examine the relationships among the problems $\Pi^{(r,s)}$ for various functions r and s. In Theorem 1, we show that if $\Pi^{(r,1)}$ is computable in time $O(2^{o(k)}p(n))$, then $\Pi^{(r,\log n)}$ is parameterized tractable. In section 3, we examine the parameterized tractability of problems in MAX SNP. In Theorem 3, we show that if some MAX SNP-hard problem Π_1 has a $O(2^{o(k)}p(n))$ parameterized algorithm, then every problem Π_2 in MAX SNP has a $O(2^{o(k)}q(n))$ parameterized algorithm. In Theorem 4, we examine the complexity of MAX C-SAT$^{(r,s)}$ for the function $r(\phi) = r'm$, where r' is a rational number and m is the number of clauses in ϕ. We show that if MAX C-SAT has a $O(2^{o(k)}p(n))$ parameterized algorithm, then MAX C-SAT$^{(r,1)}$ has a $O(2^{o(k)}q(n))$ parameterized algorithm. In section 4, we prove our main technical result, namely, that MAX C-SAT$^{(r,\log n)}$ is $W[1]$-hard. Note that many short proofs are omitted from this extended abstract.

Combining the results from sections 2, 3, and 4 gives the main result of this work. Consider the hypothesis that some MAX SNP-hard problem Π has a $O(2^{o(k)}p(n))$ parameterized algorithm. Since MAX C-SAT is in MAX SNP, this hypothesis implies that MAX C-SAT has a $O(2^{o(k)}q_1(n))$ parameterized algorithm. By Theorem 4, it follows that MAX C-SAT$^{(r,1)}$ has a $O(2^{o(k)}q_2(n))$ parameterized algorithm. By an application of Theorem 1, we have that MAX

c-Sat$^{(r,\log n)}$ is parameterized tractable. Since Max c-Sat$^{(r,\log n)}$ is $W[1]$-hard by Theorem 5, this implies that $W[1] = FPT$.

We note that earlier work by Impagliazzo, Paturi, and Zane [15] also indicates that Vertex Cover and other NP-complete problems likely do not have subexponential parameterized algorithms. In particular, their work defines a notion of completeness under "SERF" (sub-exponential reduction family) reductions for the syntactic class SNP that was originally defined by Papadimitriou and Yannakakis [18]. As shown there, the existence of a subexponential-time algorithm for any problem that is SNP-hard under SERF reductions implies that every problem in SNP has a subexponential-time algorithm. In their work, many NP-complete problems, including Vertex Cover, Independent Set, and 3-Sat, where shown to be SNP-hard under SERF reductions.

To compare our work with this earlier work, consider again the case for Vertex Cover. Since k is bounded above by n, the existence of a subexponential-time parameterized algorithm for Vertex Cover implies the existence of a subexponential-time algorithm for the usual decision version. Hence, the underlying hypothesis of our work is stronger than that of Impagliazzo, Paturi, and Zane [15]. However, our conclusion is also stronger. As shown in Corollary 17.7 of [12], if $W[1] = FPT$ then 3-SAT has a subexponential-time algorithm. Since 3-SAT is SNP-complete under SERF reductions [15], this implies that every problem in SNP has a subexponential-time algorithm. It is not known if the converse is true.

2 Preliminaries

We begin by introducing necessary concepts concerning optimization problems and the theory of parameterized complexity. For additional information, we refer readers to the comprehensive text on parameterized complexity by Downey and Fellows [12] and the classic text on NP-completeness by Garey and Johnson [14].

To begin, a parameterized problem Π is defined over the set $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet and $\mathbb{N}$ in the set of natural numbers. Therefore, each instance of the problem Π is a pair $\langle I, k\rangle$, where k is called the *parameter*. A problem Π is *parameterized tractable* if there is an algorithm running in time $O(f(k)p(|I|))$ that solves the parameterized problem Π for some polynomial p and some recursive function f. The complexity class FPT contains all parameterized tractable problems.

The theory of parameterized complexity defines a variety of reductions that preserve parameterized tractability. Here we employ the standard parameterized m-reduction. Briefly, Π_1 is *parameterized reducible* to Π_2 if there is a function $g : \mathbb{N} \to \mathbb{N}$ and $f : \Sigma^* \times \mathbb{N} \to \Sigma^*$ such that $\langle x, k\rangle \in \Pi_1 \leftrightarrow \langle f(x,k), g(k)\rangle \in \Pi_2$ and $f(x,k)$ is computable in time $g(k)p(|x|)$ for some polynomial p. Based on this reduction, a hierarchy of increasingly difficult parameterized problems, $FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots W[P]$, can be defined. This is the W-hierarchy. A problem Π is $W[t]$-hard if every problem in $W[t]$ is parameterized reducible

to Π and $W[t]$-complete if it is also in $W[t]$. Our work relies on the fact that Independent Set is $W[1]$-complete [10].

Many parameterized problems are obtained from optimization problems via parameterizations. Following the earlier work of Cai and Chen [8], we use a standard parameterization of optimization problems. For each optimization problem Π, the *standard parameterized version* of Π is to determine, given an instance I of Π and an integer k, whether the optimal solution cost $OPT_\Pi(I)$ is $\geq k$ for maximization problems or $\leq k$ in the case of a minimization problems.

As pointed out by Majahan and Raman [16], $OPT_\Pi(I)$ is always large for certain problems such as Max Cut and Max Sat. In these cases, the question of whether $OPT_\Pi(I) \geq k$ is trivial for small values of k. To overcome this difficulty, they suggest that for problems Max Sat and Max Cut, parameterized problems should be defined to determine whether $OPT(I) \geq \lceil \frac{m}{2} \rceil + k$. In this paper, this formulation is extended to $OPT_\Pi(I) \geq r(I) + ks(|I|)$, for arbitrary functions r and s. Note that the formulation of parameterized Max Sat by Majahan and Raman can be achieved by using $r(\phi) = \lceil \frac{m}{2} \rceil$, where m is the number of clauses in the boolean formula ϕ.

Definition 1. *Let Π be a maximization problem with instances I_Π and an optimal cost function OPT_Π. For functions $r : I_\Pi \to \mathbb{Q}$ and $s : \mathbb{N} \to \mathbb{Q}$, the parameterized problem $\Pi^{(r,s)}$ is defined as follows: Given an instance $\langle I, k \rangle$, determine whether $OPT_\Pi(I) \geq r(I) + ks(n)$, where $n = |I|$.*

$\Pi^{(r,s)}$ is called an extended parameterized version *of Π. When $r(n) = 0$ and $s(n) = 1$, $\Pi^{(r,s)}$ is called* the standard parameterized version *of Π. We often use Π to denote $\Pi^{(0,1)}$ when our intention is clear from the context.*

Parameterized versions of minimization problems can be defined in a similar fashion. In the literature, most parameterized tractability proofs involve explicit constructions of a solution to witness each positive answer. In particular, the following "stronger" definition of parameterized tractability was introduced in [8].

Definition 2. *Let Π be a maximization problem. The parameterized problem $\Pi^{(r,s)}$ is* parameterized tractable with witness *if there is a $O(f(k)p(|I|))$ algorithm that determines the membership of $\langle I, k \rangle$ in $\Pi^{(r,s)}$ and also produces a solution to Π that witnesses to $OPT_\Pi(I) \geq r(I)+ks(|I|)$ whenever $\langle I, k \rangle \in \Pi^{(r,s)}$, for some recursive function f and polynomial p .*

Although we primarily use the standard definition of parameterized tractability throughout this extended abstract, Definition 2 is used in section 3 to show that L reductions preserve parameterized tractability. The fact that L reductions preserve parameterized tractability with witness is crucial to our main results. Note, however, that the close relationship between search and decision means that the terms parameterized tractable and parameterized tractable with witness are equivalent in many cases. As explained in sections 4 and 5, this relationship allows us to state our main results without reference to the witness characterization.

To begin, we mention some properties of the parameterized problems $\Pi^{(r,s)}$. First note that s can be as large as $o(\log n)$ without significantly changing the parameterized complexity of $\Pi^{(r,s)}$. Consider the following technical lemma.

Lemma 1. *A parameterized problem is parameterized tractable if it is solvable in $O(2^{O(s(n)k)}p(n))$ steps for some unbounded and nondecreasing function $s(n) = o(\log n)$.*

Lemma 1 leads immediately to the following theorem.

Theorem 1. *Let $\Pi^{(r,1)}$ be a parameterized problem solvable in $O(2^{O(k)}p(n))$ steps for some p polynomial. Then for any unbounded nondecreasing function $s(n) = o(\log n)$, $\Pi^{(r,s)}$ is parameterized tractable.*

It is natural to ask whether the above theorem holds when $s = \Theta(\log n)$. As we show in section 4, this is unlikely since it implies that $W[1] = FPT$. Indeed, $\Pi^{(r,\log n)}$ appears to be parameterized intractable for certain problems. Furthermore, the parameterized intractability of $\Pi^{(r,\log n)}$ implies a strong lower bound on the running times of parameterized algorithms for $\Pi^{(r,1)}$. This is one of the keys to our overall approach.

Theorem 2. *If $\Pi^{(r,1)}$ is solvable in $O(2^{o(k)}p(n))$ steps, then $\Pi^{(r,\log n)}$ is parameterized tractable.*

3 Parameterized Tractability of MAX SNP

As noted in [12], a number of NP-complete problems such as Vertex Cover, Vertex Cover-B, Max Sat, Max c-Sat, Max k-Cut, and Independent Set for bounded degree graphs are parameterized tractable. In particular, each of these problems can be solved in time $O(2^{O(k)}p(n))$ for some polynomial p. It is natural to ask whether the running times of parameterized algorithms for these problems can be significantly improved. In this section, we work towards answering these questions through an investigation of parameterized versions of MAX SNP-hard optimization problems.

The class MAX SNP was introduced by Papadimitriou and Yannakakis [18] to study the approximability of optimization problems. As defined in [18], an optimization problem Π is in the syntactic class MAX SNP_0 if its optimal cost $OPT_\Pi(I)$ for each instance I can be expressed as $OPT_\Pi(I) = \max_S |\{v : \phi(v, I, S)\}|$, where both the instance I and the solution S are described as finite structures. The class MAX SNP contains all optimization problems that can be reduced to some problem in the class MAX SNP_0 through the following approximation-preserving reduction.

Definition 3. *[18] Let Π_1 and Π_2 be two optimization problems with cost functions f_1 and f_2. Π_1* L-reduces *to Π_2 if there are two polynomial time algorithms A and B and two constants $\alpha, \beta > 0$ such that for each instance I_1 of Π_1,*

i.) the algorithm A produces an instance $I_2 = A(I)$ such that $OPT_{\Pi_2}(I_2) \leq \alpha OPT_{\Pi_1}(I_1)$, and

ii.) given any solution S_2 for I_2 with cost $f_2(I_2, S_2)$, algorithm B produces a solution S_1 for I_1 with cost $f_1(I_1, S_1)$ such that $|OPT_{\Pi_1}(I_1) - f_1(I_1, S_1)| \leq \beta |OPT_{\Pi_2}(I_2) - f_2(I_2, S_2)|$.

It is known from the work of Cai and Chen [8] that the standard parameterized versions of all maximization problems in the MAX SNP are parameterized tractable. The proof of this earlier result shows that L reductions preserve parameterized tractability. Here we provide a more detailed account of how L reductions preserve parameterized tractability among the standard parameterized versions of optimization problems. In particular, it can be shown that L-reductions preserve subexponential-time computability.

Lemma 2. *Let Π_1 and Π_2 be two optimization problems such that Π_1 L-reduces to Π_2, and assume that the cost function for Π_2 is integer-valued. If $\Pi_2^{(0,1)}$ is solvable with witness in time $O(f(k)p(n))$ for some recursive function f and polynomial p then $\Pi_1^{(0,1)}$ can be solved in time $O(kf(O(k))q(n))$ for some q polynomial.*

Because MAX 3-SAT$^{(0,1)}$ [8,16] is parameterized tractable with witness, we obtain the following result through Lemma 2.

Corollary 1. *The standard parameterized version of each optimization problem in the class* MAX SNP *is solvable in time $O(2^{O(k)}p(n))$ for some polynomial p.*

Lemma 2 allows us to give a natural connection between the parameterized complexity of MAX SNP-hard problems and the parameterized complexity of problems in MAX SNP.

Theorem 3. *Let Π_1 be a* MAX SNP*-hard (under L-reductions) optimization problem with an integer-valued cost function. If $\Pi_1^{(0,1)}$ is solvable with witness in time $O(2^{o(k)}p(n))$ for some polynomial p, then for any optimization problem Π_2 in* MAX SNP*, $\Pi_2^{(0,1)}$ is solvable in time $O(2^{o(k)}q(n))$ for some polynomials q.*

To show that similar results hold for the extended parameterized versions of certain problems, we use the following technique that bridges the gap between the standard and extended parameterized versions of optimization problems.

Lemma 3. *Let $c > 0$ be any constant integer, let $r' \geq \frac{1}{2}$ be an any rational number, and define $r(\phi) = r' \cdot m$, where m is the number of clauses in the boolean formula ϕ. If the standard parameterized problem* MAX C-SAT *is solvable in time $O(f(k)p(n))$ for some polynomial p, then the extended parameterized problem* MAX C-SAT$^{(r,1)}$ *is solvable in time $O(f(11k+8)q(n))$, for some polynomial q.*

Proof. Assume that there is an algorithm A solving the parameterized problem MAX C-SAT as stated. We describe an algorithm B that solves MAX C-SAT$^{(r,1)}$

by calling the algorithm A. The algorithm B uses the approach found in Proposition 8 and Theorem 9 of Mahajan and Raman [16].

Let F be a set of clauses, and let $\langle F, k\rangle$ be a given instance for MAX C-SAT$^{(r,1)}$. The algorithm B operates as follows.

Input $\langle F, k\rangle$.
Let U be the set of unit clauses in F.
(1) While U contains clauses of the form (x) and $(\neg x)$,
remove both clauses and reduce k by 1.
(2) If $|U| \geq r'm + k$ return "YES"
(3) If $\frac{m}{2} + \frac{|F-U|}{4} - 1 \geq r'm + k$ return "YES"
(4) Otherwise, call algorithm A on input $\langle F, r'm + k\rangle$ and return its answer.

To see that this algorithm correctly solves MAX C-SAT$^{(r,1)}$ on input $\langle F, k\rangle$, first consider the set U of unit clauses in F. If U contains two clauses of the form (x) and $(\neg x)$, both can be removed since any truth assignment of x satisfies exactly one of these two clauses. In this case, the value k can be reduced by 1. If U contains no such pair of clauses, then all the clauses in U can be satisfied simultaneously. Hence, if $|U| \geq r'm + k$, there is an assignment to the variable satisfying at least $r'm + k$ clauses and the algorithm B correctly answers yes. Furthermore, by Proposition 8 of [16], there exists a satisfying assignment of F that satisfies at least $\lceil \frac{m}{2} \rceil + \frac{|F-U|}{4} - 1$ clauses in F. Hence, if $\frac{m}{2} + \frac{|F-U|}{4} - 1 \geq r'm + k$, at least $r'm + k$ clauses of F can be satisfied simultaneously. So, the algorithm B also answers correctly in this case. In all other cases, algorithm B calls algorithm A on input $\langle F, r'm + k\rangle$. Since algorithm A correctly solves MAX C-SAT$^{(0,1)}$, algorithm B is also correct.

It is easy to see that steps (1)–(3) of algorithm B can be performed in a polynomial number of steps. Step (4) involves a call to algorithm A on input $\langle F, r'm + k\rangle$. Since $\frac{m}{2} + \frac{|F-U|}{4} - 1 < r'm + k$, we have that $|F - U| \leq 4(r - \frac{1}{2})m + 4k + 4$. Since $|U| < r'm + k$, we have that $m = |U| + |F - U| \leq rm + k + 4(r - \frac{1}{2}) + 4k + 4$. Therefore, we have that the number of clauses in F is bounded a linear function of k[1], i.e.,

$$m \leq \frac{5k + 4}{3 - 5r}.$$

Because $r' \geq \frac{1}{2}$, it follows that $m \leq 10k + 8$ and $k' = r'm + k \leq 10r'k + 8r' + k$. Substituting this new value of k into the running time of the algorithm A gives the required running time for the algorithm B.

Lemma 3 immediately gives the following theorem.

Theorem 4. *Let $c > 0$ be any constant integer, let $r' \geq \frac{1}{2}$ be an any rational number, and define $r(\phi) = r' \cdot m$, where m is the number of clauses in the boolean formula ϕ. If the parameterized problem* MAX C-SAT$^{(0,1)}$ *is solvable in time $O(2^{o(k)}p(n))$ for some p polynomial, then* MAX C-SAT$^{(r,1)}$ *is solvable in time $O(2^{o(k)}q(n))$, for some polynomial q.* □

[1] This also suggests that step (4) will only be executed when $r < \frac{3}{5}$.

4 Parameterized Complexity of MAX SNP-Hard Problems

In contrast to our results in the previous section, we now show that for some optimization problem Π, the parameterized version $\Pi^{(r,\log n)}$ may not be parameterized tractable.

Theorem 5. *Let r' be a rational constant such that $\frac{1}{2} < r' < 1$ and define $r(\phi) = r'm$, where m is the number of clauses in ϕ. Then for any natural number $c \geq 3$,* MAX c-SAT$^{(r,\log n)}$ *is $W[1]$-hard.*

Proof. It suffices to show that the $W[1]$-hard problem INDEPENDENT SET can be transformed to MAX c-SAT$^{(r,\log n)}$ through a standard parameterized m reduction.

The parameterized problem INDEPENDENT SET is defined as follows. Given a graph $G = (V, E)$ of n nodes and an integer k, determine whether there is a subset V' of V of size k in G such that no two vertices in V' are connected by an edge in G. We describe a process to transform $\langle G, k\rangle$ to an instance for problem MAX c-SAT$^{(r,\log n)}$ for some $r' \geq \frac{1}{2}$. The reduction consists of the following five steps.

Step 1. Construct an anti-monotonic Boolean circuit C_1 from G as follows. Let $V = \{v_1, \cdots, v_n\}$. The circuit C_1 consists of n input variables $X = (x_1, \cdots, x_n)$, an AND gate as the output, and $|E|$ intermediate OR gates, each of which has the output wired to an input of the AND gate. For each edge $e = (v_i, v_j)$ in G, an OR gate g_e is constructed with two inputs $\neg x_i$ and $\neg x_j$. By associating a setting of the variables $x_1, \ldots, x_n$ with a subset of V in the natural way, it is straightforward to verify that C_1 has a satisfying assignment of weight k if and only if G has an independent set of size k.

Step 2. Convert the circuit C_1 into another anti-monotonic circuit C_2 that has nk input variables. These nk variables are organized into k blocks of n. Let $Y = [(y_1^{(1)}, \cdots y_n^{(1)}), (y_1^{(2)}, \cdots y_n^{(2)}), \cdots, (y_1^{(k)}, \cdots y_n^{(k)})]$ be this set of input variables. As with C_1, we have an AND gate as output. In addition, the circuit C_2 contains three sets of OR gates, E_1, E_2, and E_3, defined as follows.

$$E_1 = \{\, (\neg y_i^{(t)} \vee \neg y_j^{(t)}) \,:\, 1 \leq t \leq k,\, 1 \leq i < j \leq n \,\},$$

$$E_2 = \{\, (\neg y_i^{(s)} \vee \neg y_j^{(t)}) \,:\, \text{for each gate } \neg x_i \vee \neg x_j \in C_1,\, 1 \leq i, j \leq n, 1 \leq s, t \leq k\},$$

$$\text{and } E_3 = \{\, (\neg y_i^{(s)} \vee \neg y_i^{(t)}) \,:\, 1 \leq s < t \leq k, 1 \leq i \leq n \,\}.$$

Each of the OR gates has an output wired to an input of the AND gate.

Notice that the three sets of OR gates enforce specific conditions. The set E_1 enforces the condition that no more than one variable in each block of n can be set to true. The set E_3 enforces the condition that no more than one variable in position j ($y_j^{(t)}$) of some block is set to true. Sets E_1 and E_3 force any satisfying assignment to contain at most k variable that are set to true, with at most one coming from each block of n and each position j. Intuitively, the ith variable in

a block being set to true will correspond to the vertex i being in an independent set V'.

It is easy to show that C_2 has a weight k satisfying assignment if and only if C_1 has a weight k satisfying assignment.

Step 3. Transform the anti-monotonic circuit C_2 into a monotonic circuit C_3 that contains $2k \log n$ input variables, organized into k blocks of size $2 \log n$. Let $Z = (z^{(1)}, \cdots, z^{(k)})$ be the k blocks of input variables, where for each $t = 1, \cdots, k$, $z^{(t)} = (u_1^{(t)}, v_1^{(t)}, \cdots, u_{\log n}^{(t)}, v_{\log n}^{(t)})$ is a vector of $2\log n$ variables. In this construction, $z^{(t)}$ corresponds to the tth block in Y. This requires some explanation.

A desired assignment to Y in C_2 will have exactly one variable assigned to true in each block. For each block t of Y, the variable that is assigned to true, say $y_i^{(t)}$, can be specified by its position i within the block. When $1 \leq i \leq n$, the position i can be encoded by a binary number B of length $\log n$. Let $B = b_1^{(t)} \cdots b_{\log n}^{(t)}$ with each $b_l^{(t)} \in \{0, 1\}$. In C_3, each bit $b_l^{(t)}$ is encoded by a pair of variables $u_l^{(t)}, v_l^{(t)}$. For $l = 1, \cdots, k$,

(1) $b_l^{(t)} = 1$ if and only if $u_l^{(t)} = 1$ and $v_l^{(t)} = 0$, and
(2) $b_l^{(t)} = 0$ if and only if $u_l^{(t)} = 0$ and $v_l^{(t)} = 1$.

Notice that each input variable in $y_i^{(t)}$ can be represented by an AND of input variables from Z. Let gate $g_i^{(t)}$ in C_3 represent the input variable $y_i^{(t)}$ in Y, where $g_i^{(t)} = \bigwedge_{l=1}^{\log n} w_l^{(t)}$ with $w_l^{(t)} = u_l^{(t)}$ if $b_l^{(t)} = 1$ in $B = i$ and $w_l^{(t)} = v_l^{(t)}$ if $b_l^{(t)} = 0$ in $B = i$. Since C_2 is an anti-monotonic circuit, we only need to represent the negation of an input variable $y_i^{(t)}$ in Y. For this purpose, we use $\bar{g}_i^{(t)} = \neg g_i^{(t)} = \bigvee_{l=1}^{\log n} w_l^{(t)}$, where $w_l^{(t)} = u_l^{(t)}$ if $b_l(t) = 0$ in $B = i$ and $w_l^{(t)} = v_l^{(t)}$ if $b_l(t) = 1$ in $B = i$. It is not hard to verify the correctness of this representation.

Continuing the construction, each OR gate $\neg y_i^{(s)} \vee \neg y_j^{(t)}$ in circuit C_2 is represented by a gate $\bar{g}_i^{(s)} \vee \bar{g}_j^{(t)}$. This is an OR of $2 \log n$ variables in Z. Additionally, we need to guarantee that each pair of input variables $u_l^{(t)}, v_l^{(t)}$ always take exclusive values in a desired assignment. To enforce this, we introduce a set H of gates, where $H = \{\, h_l^{(t)} = u_l^{(t)} \vee v_l^{(t)} \,:\, l = 1, \cdots, \log n, t = 1, \cdots, k \,\}$. The gates in H force at least one variable from each pair $u_l^{(t)}$, $v_l^{(t)}$ to evaluate to true. Since there are exactly $2k \log n$ variables, a weight $k \log n$ satisfying assignment causes exactly one of $u_l^{(t)}$, $v_l^{(t)}$ to be true.

It is straightforward to show that C_3 has a weight $k \log n$ satisfying assignment if and only if C_2 has a weight k satisfying assignment.

Step 4. Reformulate the weighted satisfiability problem for the monotone circuit C_3 into a parameterized MAX C-SAT$^{(r, \log n)}$ problem. From step 3, C_3 is a monotonic circuit with $2k \log n$ input variables that is an AND of ORs. Note that all of the OR gates in C_3 either have fan-in 2 or fan-in $s = 2 \log n$. We now build a boolean formula in CNF for C_3.

Define

$$F_1 = \{C_g = (w_1^{(g)}, \cdots, w_s^{(g)}) : \text{ gate } g = \bigvee_{i=1}^{s} w_i^{(g)} \text{ is in } C_3 \},$$

$$F_2 = \{(w) : \ w \text{ is an input variable of } C_3\}, \text{ and}$$

$$H' = \{(u, v), (\neg u, \neg v) : \ u, v \text{ are two paired input variables of } C_3 \}.$$

Furthermore, let F_3 be the set containing $(2k \log n + 1)$ copies of each clause in H'. Let $F = F_1 \cup F_2 \cup F_3$. Note that $|F_2| = 2k \log n$.

If C_3 has a weight $k \log n$ satisfying assignment, then there is an assignment of the variables of F that satisfies $|F_1| + |F_3| + k \log n$ clauses in F. Similarly, if F has an assignment that satisfies $|F_1| + |F_3| + k \log n$, then all the clauses in F_3 must evaluate to true. If not, then at least $2k \log n + 1$ clauses in F_3 evaluate to false. This is a contradiction. Moreover, if all the clauses in F_3 evaluate to true, then exactly $k \log n$ variables are set to true. Hence, all the clauses in F_1 must evaluate to true. Therefore, C_3 has a weight $k \log n$ satisfying assignment.

To complete the conversion to a formula in c-CNF, it suffices to convert all the large ORs in F_1 to clauses of size c. This can be done using additional variables as in the standard reduction from SAT to 3-SAT [3, p.438]. If these new clauses are placed into F_1, then, as verified in the previous paragraph, C_3 has a weight $k \log n$ satisfying assignment if and only if F has an assignment satisfying $|F_1| + |F_3| + k \log n$ clauses.

Step 5. Let $N = |F_1| + |F_3|$. Note that the total number of clauses in F is $N + 2k \log n$, where n is the number of vertices in the original graph G. We next pad some new unit clauses into F so that there exists an assignment to the variables satisfying $r'm + k \log n$ clauses if and only if G has an independent set of size k, where m is the number of clauses.

Now, add M new variables and add one unit clause to F for each new variable and its negation. This new formula F has $m = 2M + N + 2k \log n$ clauses, and there exists an assignment to the variables satisfying $M + N + k \log n$ clauses if and only if G has an independent set of size k. It suffices to show that we can pick a value for M such that $r'm + k \log n = M + N + k \log n$.

The appropriate value for M must satisfy $r' = (M + k \log n + N)/(2M + 2k \log n + N)$. We can rewrite this as $r' = 1 - \frac{M + k \log n}{2(M + k \log n) + N}$, and hence $M = \frac{(1 - r')N - r' k \log n}{2r' - 1}$. Because $N >> k \log n$, such an M exists for any $\frac{1}{2} < r' < 1$. Moreover, we can compute M from r', N, k, and $\log n$ and produce the correct number of unit clauses.

An adjustment must be made to the factor $\log n$. It can be verified that N is a polynomial in n. Moreover, M is linear in N. So $\log m = O(\log N) = O(\log n)$. Therefore, we can add $2dk \log n$ unit clauses to F, for some constant d, so that exactly $rm + k \log m$ clauses can be satisfied. This completes the reduction.

Finally, note that the reduction takes an instance $\langle G, k \rangle$ of Independent Set and produces an instance $\langle F, k \rangle$ of Max c-Sat$^{(r, \log n)}$. Since this is a parameterized m-reduction, it follows that Max c-Sat$^{(r, \log n)}$ is $W[1]$-hard.

Corollary 2. *Let r' be a rational constant such that $\frac{1}{2} < r' < 1$, and define $r(\phi) = r'm$, where m is the number of clauses in ϕ. The problem* MAX SAT$^{(r,\log n)}$ *is $W[1]$-hard.*

Theorem 5 completes the technical results leading up to our main result.

Theorem 6. *Let Π be a* MAX SNP*-hard optimization problem with an integer-valued cost function. The standard parameterized version of Π cannot be solved with witness in time $O(2^{o(k)}p(n))$ for any polynomial $p(n)$ unless $W[1] = FPT$.*

Proof. Assume that for some MAX SNP-hard optimization problem Π, its standard parameterized version $\Pi^{(0,1)}$ is solvable with witness in time $O(2^{o(k)}p(n))$ for some polynomial $p(n)$. Then by Theorem 3, MAX C-SAT$^{(0,1)}$ is solvable in time $O(2^{o(k)}q(n))$, for some polynomial $q(n)$. By Theorem 4, a $O(k2^{o(k)}q(n))$ algorithm exists for MAX C-SAT$^{(r,1)}$ for any r. By Theorem 2, MAX C-SAT$^{(r,\log n)}$ is parameterized tractable. Together with Theorem 5, this implies $W[1] = FPT$.

Since the proof of Theorem 6 relies on Theorem 3, it does not appear that we can easily remove the word "witness" from the statement of our main result. However, in practice, it is often the case that the complexities of decision problems and their witness versions are closely related. In the case of VERTEX COVER, it is easy to show that VERTEX COVER is solvable in time $O(2^{o(k)}p(n))$ if and only if it is solvable with witness in time $O(2^{o(k)}q(n))$ for polynomials p and q. Hence, Theorem 6 gives the following immediate corollary.

Corollary 3. *The parameterized problems* MAX SAT*,* MAX C-SAT*,* VERTEX COVER*,* VERTEX COVER*-B,* INDEPENDENT SET*-B,* DOMINATING SET*-B, and* MAX C-CUT *cannot be solved in time $O(2^{o(k)}p(n))$ for any polynomial $p(n)$ unless $W[1] = FPT$.* □.

5 Conclusion

Our main results provide a simple framework for proving strong lower bounds on the parameterized complexity of problems within FPT. To achieve a $2^{o(k)}p(n)$ lower bound, it suffices to prove that a problem is MAX SNP-hard and that the witness version nicely reduces to the decision version. As mentioned by Bellare and Goldwasser [6], it is well-known that search is polynomial-time Turing reducible to decision for every NP-complete problem. To obtain Corollary 3, we require a more restrictive notion of reducibility between the witness and decision versions of parameterized problems. In particular, we require that the reduction between the witness and decision version does not greatly increase the value of the parameter k. It is not immediately obvious that search reduces to decision for every NP-complete problem when this requirement is added. Nevertheless, it is the case that the witness version reduces to the decision version in this way for many NP-complete problems, such as those mentioned in Corollary 3.

More generally, our techniques provide a framework for relating the complexities of various parameterizations of the same problem. We believe that this framework may lead to lower bounds on non MAX SNP-hard problems as well.

References

1. J. Alber, H. Bodlaender, H. Fernau, and R. Niedermeier. Fixed parameter algorithms for planar dominating set and related problems. In *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory (SWAT 2000)*, volume 1851 of *Lecture Notes in Computer Science*, pages 97–110. Springer-Verlag, 2000.
2. J. Alber, J. Gramm, and R. Niedermeier. Faster exact algorithms for hard problems: A parameterized point of view. *Discrete Mathematics*, 2001. to appear.
3. S. Arora and C. Lund. Hardness of approximations. In Dorit Hochbaum, editor, *Approximation Algorithms for NP-hard problems*, pages 399–446. PWS Publishing, 1997.
4. B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41:153–180, 1994.
5. R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, 65:163–168, 1998.
6. M. Bellare and S. Goldwasser. The complexity of decision versus search. *SIAM Journal on Computing*, 23(1):97–119, February 1994.
7. S. Buss, 1989. Personal Communication with Downey and Fellows cited in [12, p.5].
8. L. Cai and J. Chen. On fixed-parameter tractability and approximability of NP optimization problems. *Journal of Computer and System Sciences*, 54(3):465–474, June 1997.
9. J. Chen, I. A. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. In *Proceedings of the 25th International Workshop on Graph-Theoretical Concepts in Computer Science*, volume 1665 of *Lecture Notes in Computer Science*, pages 313–324. Springer-Verlag, 1999.
10. R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: On completeness for $W[1]$. *Theoretical Computer Science*, 141:109–131, 1995.
11. R. G. Downey and M. R. Fellows. Parameterized computational feasibility. In *Proceedings of Feasible Mathematics II*, pages 219–244. Birkhauser, 1995.
12. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, New York, 1999.
13. R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics: From DIMACS to DIMATIA to the Future*, volume 49 of *AMS-DIMACS Proceeding Series*, pages 49–99. AMS, 1999.
14. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, San Francisco, 1979.
15. R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? In *Proceedings of the 39th Symposium on Foundations of Computer Science*, pages 653–664. IEEE Computer Society Press, 1998.
16. M. Mahajan and V. Raman. Parameterizing above guaranteed values:MaxSat and MaxCut. *Journal of Algorithms*, 31:335–354, 1999.
17. R. Niedermeier and P. Rossmanith. Upper bounds for vertex cover further improved. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 561–570. Springer-Verlag, 1999.
18. C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and Systems Sciences*, 43:425–440, 1991.

Improved Lower Bounds on the Randomized Complexity of Graph Properties*

Amit Chakrabarti and Subhash Khot

Department of Computer Science, Princeton University
35 Olden Street, Princeton NJ 08544, USA.
{amitc,khot}@cs.princeton.edu
FAX: +1-(609)-258-1771

Abstract. We prove a lower bound of $\Omega(n^{4/3}\log^{1/3} n)$ on the randomized decision tree complexity of any nontrivial monotone n-vertex bipartite graph property, thereby improving the previous bound of $\Omega(n^{4/3})$ due to Hajnal [H91]. Our proof works by improving a probabilistic argument in that paper, which also improves a graph packing lemma proved there. By a result of Gröger [G92] our complexity lower bound carries over from bipartite to general monotone n-vertex graph properties. Graph packing being a well-studied subject in its own right, our improved packing lemma and the probabilistic technique used to prove it, may be of independent interest.

Keywords: Decision tree complexity, monotone graph properties, randomized complexity, randomized algorithms, graph packing, probabilistic method.

1 Introduction

Consider the problem of deciding whether or not a given input graph G has a certain (isomorphism invariant) property P. The graph is given by an oracle which answers queries of the form "is (x, y) an edge of G?" A *decision tree algorithm* for P makes a sequence of such queries to the oracle, where each query may depend upon the information obtained from the previous ones, until sufficient information about G has been obtained to decide whether or not P holds for G, whereupon it either accepts or rejects. Let $\mathcal{A}_P$ denote the set of decision tree algorithms for P and for $A \in \mathcal{A}_P$, let $\text{cost}(A, G)$ denote the number of queries that A asks on input G. The quantity $\mathcal{C}(P) = \min_A \max_G \text{cost}(A, G)$ is called the *deterministic decision tree complexity*, or simply the deterministic complexity of P.

A *randomized decision tree algorithm* for P is a probability distribution $\mathcal{D}$ over $\mathcal{A}_P$, and its cost (on input G) is the expectation of $\text{cost}(A, G)$ with A drawn from $\mathcal{D}$:

$$\text{cost}^R(\mathcal{D}, G) = \sum_{A \in \mathcal{A}_P} \Pr_{\mathcal{D}}[A]\,\text{cost}(A, G) \ .$$

* This work was supported in part by NSF Grant CCR-96-23768, ARO Grant DAAH04-96-1-0181, and NEC Research Institute.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 285–296, 2001.

The *randomized decision tree complexity*, or simply the randomized complexity, of P is defined to be

$$\mathcal{C}^R(P) = \min_{\mathcal{D}} \max_{G} \text{cost}^R(\mathcal{D}, G) \ .$$

An n-vertex graph property is said to be *nontrivial* if there is at least one n-vertex graph which has the property and at least one which does not. It is said to be *monotone* if addition of edges does not destroy the property. Let $\mathcal{P}_n$ denote the set of all nontrivial monotone n-vertex graph properties. We shall need to consider the analogously defined notion of randomized complexity for *bipartite* graph properties; let $\mathcal{P}_{n,n}$ denote the set of all nontrivial monotone (n, n)-bipartite graph properties.

Clearly any n-vertex graph property P satisfies $\mathcal{C}^R(P) \leq \mathcal{C}(P) \leq \binom{n}{2} = O(n^2)$. A classic result of Rivest and Vuillemin [RV76] shows that any $P \in \mathcal{P}_n$ has $\mathcal{C}(P) = \Omega(n^2)$, which settles the deterministic complexity of monotone properties up to a constant.[1] We remark that monotonicity is crucial for this result; there are examples of nontrivial non-monotone graph properties with $\mathcal{C}(P) = O(n)$.

The situation is far less satisfactory for randomized complexity. The first nonlinear lower bound on $\mathcal{C}^R(P)$, for general $P \in \mathcal{P}_n$, was an $\Omega(n \log^{1/12} n)$ bound proven by Yao [Y87]. This was subsequently improved by Valerie King [K88] to $\Omega(n^{5/4})$ and later by Hajnal [H91] to $\Omega(n^{4/3})$. The only other significant work in the area is due to Gröger [G92] who established lower bounds stronger than Hajnal's for certain special classes of graph properties.

No property in $\mathcal{P}_n$ is known to have randomized complexity below $n^2/4$. Closing this gap between Hajnal's lower bound and this upper bound is one of the most important open problems concerning the complexity of graph properties. It is remarkable that this quarter-century-old problem has yielded so few results.

In this paper we take a small step by proving

Theorem 1.1 (Main Theorem). *Any property $P \in \mathcal{P}_n$ satisfies $\mathcal{C}^R(P) = \Omega(n^{4/3} \log^{1/3} n)$.* [2]

Our proof will rely on an important theorem from the pioneering work of Yao [Y87], as well as on a framework developed by Hajnal [H91]. In this framework we associate with a graph property a special pair of graphs which cannot be "packed" together. We then argue that if the property has low randomized complexity, then certain degree upper bounds can be proven for these special graphs. Finally, we use these degree bounds to prove that the special graphs can be "packed", thereby arriving at a contradiction.

The notion of graph packing, which we shall formally define later, is a well-studied subject in its own right [B78]. A packing lemma (Lemma 2.8) which we

[1] However, in the world of deterministic complexity, a far more interesting conjecture is that any $P \in \mathcal{P}_n$ has $\mathcal{C}(P) = \binom{n}{2}$ exactly. Remarkably, this conjecture remains open to this day.

[2] Throughout this paper $\log x$ denotes the logarithm of x to the base 2. The natural logarithm of x is denoted by $\ln x$.

establish in this paper is therefore of independent interest since it improves a packing theorem due to Hajnal and Szegedy [HS92].

The rest of the paper is organized as follows: in Section 2, we define some preliminary notions, describe the framework alluded to above and prove Theorem 1.1 assuming our graph packing lemma. In Section 3 we state and prove a technical lemma which is then used in Section 4 to prove the packing lemma we need, thereby completing the proof of Theorem 1.1. We conclude with some remarks in Section 5.

2 Preliminaries and Proof Outline

The first important step is to change the objects of study from graph properties to *bipartite* graph properties. A result of Gröger [G92] lets us do just that.

Theorem 2.1. *Let $f(n)$ be a function satisfying $f(n) = O(n^{3/2})$ and suppose any $P \in \mathcal{P}_{n,n}$ satisfies $\mathcal{C}^R(P) = \Omega(f(n))$. Then any $Q \in \mathcal{P}_n$ satisfies $\mathcal{C}^R(Q) = \Omega(f(n))$.*

Proof. This is a simple restatement of Theorem 3.5 of [G92]. □

For the purposes of proving a lower bound of $\Omega(n^{4/3}\log^{1/3} n)$, we may therefore safely concentrate on monotone bipartite graph properties alone. We now need some definitions.

Definition 2.2 (Basic definitions). *An (m,n)-bipartite graph G is a graph whose vertices can be partitioned into two independent sets, denoted $V_L(G)$ and $V_R(G)$ respectively, of sizes m and n respectively. The edge set of G is denoted $E(G)$. For such a graph we define*

$$\Delta_L(G) = \max_{v \in V_L(G)} \deg_G(v) \ , \qquad \delta_L(G) = \frac{1}{|V_L(G)|} \sum_{v \in V_L(G)} \deg_G(v) \ = \frac{|E(G)|}{|V_L(G)|} \ .$$

$\Delta_R(G)$ and $\delta_R(G)$ are defined similarly. When $|V_L(G)| = |V_R(G)|$ we define $\delta(G) = \delta_L(G) = \delta_R(G)$. We define $\bar{G}$ to be the (m,n)-bipartite graph with the same bipartition and with edge set $V_L(G) \times V_R(G) - E(G)$.

Definition 2.3 (Sparseness). *The bipartite graph G is said to be* L-sparse *if $V_L(G)$ contains at least $\frac{1}{2}|V_L(G)|$ isolated vertices, i.e. vertices of degree 0. The notion of* R-sparseness *is defined analogously.*

Let $P \in \mathcal{P}_{n,n}$. An (n,n)-bipartite graph G is called a *minterm* of P if G satisfies P but removing any edge from G yields a graph which does not. Suppose we associate with G an n-tuple $(d_1, d_2, \ldots, d_n)$ with $d_1 \geq \ldots \geq d_n$ where the d_i are the degrees of the vertices in $V_L(G)$; we then say that G is an *L-first minterm* of P if it is a minterm and its associated n-tuple is lexicographically smallest amongst all minterms. We say that G is an *L-first sparse minterm* of

P if it is a minterm, is L-sparse and its associated n-tuple is lexicographically smallest amongst all L-sparse minterms. We define *R-first minterms* and *R-first sparse minterms* analogously.

Finally, we define the *dual* of a property $P \in \mathcal{P}_{n,n}$ to be the property $P^* \in \mathcal{P}_{n,n}$ such that a graph G satisfies P^* iff $\bar{G}$ does not satisfy P.

Lemma 2.4. *For any $P \in \mathcal{P}_{n,n}$ either P or P^* has an R-sparse minterm.*

Proof. Let G be an edge-maximal R-sparse (n,n)-bipartite graph. Then $\bar{G}$ is isomorphic to G; therefore G must satisfy either P or P^*. □

It is easy to see that any decision tree algorithm for P can be converted into one for P^*; this gives $\mathcal{C}^R(P) = \mathcal{C}^R(P^*)$. Therefore from now on we shall assume WLOG that G has an R-sparse minterm. The next theorem summarizes the key result of Yao [Y87] and an extension of the result due to Hajnal [H91].

Theorem 2.5 ([Y87,H91]). *For $P \in \mathcal{P}_{n,n}$, the following hold*
(1) *If G is a minterm of P then $\mathcal{C}^R(P) = \Omega(|E(G)|)$.*
(2) *If G is either an L-first minterm or an L-first sparse minterm, then*

$$\mathcal{C}^R(P) = \Omega(n\Delta_L(G)/\delta_L(G)) \ ,$$

and a similar statement holds for R-first minterms and R-first sparse minterms. □

2.1 Graph Packing

We now introduce the key graph theoretic concept which we shall need. Let us say that graphs G and H *can be packed* if there is a way to identify their vertices without identifying any edge of G with an edge of H. Such an identification, when it exists, shall be called a *packing* of G and H. To see the relevance of this concept, consider the case when G and H are minterms of P and P^* respectively, for some property $P \in \mathcal{P}_{n,n}$. To say that G and H can be packed is equivalent to saying that G is isomorphic to a subgraph of $\bar{H}$. Now from monotonicity and the definition of dual properties one can see that this gives rise to a contradiction.

These ideas are formalized in the next definition[3] and the following theorem.

Definition 2.6. *Let G and H be (m,n)-bipartite graphs. A* packing *of G and H is a pair of bijections $\varphi_L : V_L(G) \to V_L(H)$ and $\varphi_R : V_R(G) \to V_R(H)$ such that for any $x \in V_L(G)$ and $y \in V_R(G)$, either $(x,y) \notin E(G)$ or $(\varphi_L(x), \varphi_R(y)) \notin E(H)$. We say that G and H* can be packed *if there exists such a packing.*

Theorem 2.7 ([Y87]). *For $P \in \mathcal{P}_{n,n}$, let G be a minterm of P and H be a minterm of P^*. Then G and H cannot be packed.* □

[3] We have defined the notion of packing only for bipartite graphs here because that is all we need. In the literature, packing has been studied both for general graphs as well as bipartite graphs.

2.2 Outline of the Proof of the Main Theorem

We are now ready to outline the proof of Theorem 1.1. Let $P \in \mathcal{P}_{n,n}$ and let $q = q(n)$ be a parameter to be fixed later. We wish to prove that $\mathcal{C}^R(P) = \Omega(nq)$. Suppose this is not the case. Let G be an R-first sparse minterm of P and H be an L-first minterm of P^*. By part (1) of Theorem 2.5 the following conditions hold:

$$\delta(G) \leq q \ , \qquad \delta(H) \leq q \ .$$

Using these in part (2) of Theorem 2.5 gives us the following additional conditions:

$$\Delta_R(G) \leq q^2 \ , \qquad \Delta_L(H) \leq q^2 \ .$$

What we would like to show is that for an appropriate choice of q, these conditions imply that G and H can be packed. Then by Theorem 2.7 we would have a contradiction.

The above framework is the same as that used by Hajnal [H91]. Our improvement is in the parameters of the packing lemma. Our improved lemma says:

Lemma 2.8 (Packing Lemma). *Set $q = (\frac{1}{16}\varepsilon n \log n)^{1/3}$. Let G and H be (n,n)-bipartite graphs with $\delta(G) \leq q$, $\delta(H) \leq q$, $\Delta_R(G) \leq q^2$ and $\Delta_L(H) \leq q^2$. Furthermore, suppose G is R-sparse. Then if ε is a small enough constant, G and H can be packed.*

Remark. This is a stronger result than that of Hajnal and Szegedy [HS92]; this makes the lemma interesting on its own.

As outlined above, proving this lemma will establish that $\mathcal{C}^R(P) = \Omega(nq) = \Omega(n^{4/3} \log^{1/3} n)$ with the above choice of q. This will prove the Main Theorem 1.1. The rest of the paper will therefore be devoted to proving this lemma.

3 A Technical Lemma

The proof of our improved packing lemma will depend on a probabilistic fact: specifically, a tail inequality. We shall be interested in considering set systems with the property that a small random sample of the ground set is unlikely to hit too many of the sets in the system. More precisely, the property is that the number of sets *missed* by the sample is only a constant factor below the expected number, with high probability. The purpose of this section is to establish that certain simple conditions, if satisfied by the set system, guarantee this type of property. As it turns out, it suffices to upper bound the size of each set, and the maximum and average number of sets containing each element of the ground set; of course we also need a large enough set system over a large enough ground set. We shall call a set system *favourable* — with the appropriate parameters — if it satisfies these conditions.

Assumption 3.1 *Throughout this section we assume that n is large enough and that $r(n)$, $s(n)$ and $t(n)$ are integer valued functions in $o(n) \cap \omega(1)$.*

Definition 3.2. *Let V be a finite set and let $\mathcal{F} \subseteq 2^V$ be a set system on ground set V. We say that $\mathcal{F}$ is $(n, r(n), s(n), \bar{s}(n))$-*favourable *if*

$$|V| = n \ , \qquad |\mathcal{F}| \geq n \ ,$$

$$\forall F \in \mathcal{F} \quad |F| \leq r(n) \ ,$$

$$\forall v \in V \quad |\{F \in \mathcal{F} : \ v \in F\}| \leq s(n) \ , \quad \text{and}$$

$$\frac{1}{n} \sum_{v \in V} |\{F \in \mathcal{F} : \ v \in F\}| \leq \bar{s}(n) \ .$$

Now consider a set system $\mathcal{F}$ on ground set V, and a function $t(n)$. Let S be a random subset of V of size $t(n)$, chosen uniformly from all subsets of size $t(n)$. Our Technical Lemma is concerned with the behaviour of the following random variable:

$$X(\mathcal{F}; t(n)) = |\{F \in \mathcal{F} : \ F \cap S = \emptyset\}| \ . \tag{1}$$

Lemma 3.3 (Technical Lemma). *Let $\mathcal{F}$ be $(n, r(n), s(n), \bar{s}(n))$-favourable. Suppose $r(n)t(n) \leq \frac{1}{4}\varepsilon n \log n$, and $t(n)s(n)\bar{s}(n) \leq n^{2-3\varepsilon}$ for some constant $\varepsilon > 0$. Then we have*

$$\Pr\left[X(\mathcal{F}; t(n)) < \frac{1}{2} n^{1-\varepsilon}\right] \leq \frac{1}{n^2} \ .$$

Example. It may help to first think about a concrete example of a favourable set system and what the lemma says about it. Consider the ground set $V = \{1, 2, \ldots, n\}$ and let $\mathcal{F}$ be the collection of all n intervals of size $3n^{1/4}$ (say) with wrap-around (i.e. n and 1 are consecutive). Every interval is of size $3n^{1/4}$ and each point of the ground set belongs to $6n^{1/4}$ intervals. Therefore this $\mathcal{F}$ is $(n, 3n^{1/4}, 6n^{1/4}, 6n^{1/4})$-favourable. A straightforward calculation shows that a random subset of V of size $5n^{3/4}$ (say) is expected to be disjoint from $\Omega(n)$ of these intervals. From the above lemma we can conclude, in particular, that it is disjoint from $\Omega(n^{0.99})$ intervals, with "high" probability.

In order to prove Lemma 3.3, we define another random variable Y which "behaves like" $X(\mathcal{F}; t(n))$ and is easier to handle. Let us number the elements of V from 1 to n and set $p = 2t(n)/n$. Let $Z_1, \ldots, Z_n$ be i.i.d. boolean random variables with $\Pr[Z_i = 1] = p$. Let $S' \subseteq V$ be the random subset given by $S' = \{i : \ Z_i = 1\}$ and define

$$Y = |\{F \in \mathcal{F} : \ F \cap S' = \emptyset\}| \ . \tag{2}$$

The next lemma connects Y with X.

Lemma 3.4. *For any α we have* $\Pr[X(\mathcal{F}; t(n)) < \alpha] \leq 2 \cdot \Pr[Y < \alpha]$.

Proof. We proceed as in [H91]. For $0 \leq k \leq n$ let $\pi_k = \Pr[X(\mathcal{F}; k) < \alpha]$. Observe that $\pi_0 \leq \pi_1 \leq \cdots \leq \pi_n$. Let $A = \lfloor \frac{1}{2}np \rfloor$ and $B = \lfloor \frac{3}{2}np \rfloor$. We have

$$\Pr[Y < \alpha] = \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} \pi_k \geq \pi_A \sum_{k=A}^{B} \binom{n}{k} p^k (1-p)^{n-k} \geq \frac{1}{2}\pi_A \ .$$

Noting that $A = t(n)$ completes the proof. □

Lemma 3.5. *Under the hypotheses of the Technical Lemma,* $E[Y] \geq n^{1-\varepsilon}$.

Proof. We have $E[Y] = \sum_{F \in \mathcal{F}} (1-p)^{|F|} \geq n(1-p)^{r(n)}$ using the fact that $\mathcal{F}$ is $(n, r(n), s(n), \bar{s}(n))$-favourable. For any constant $\alpha > 1$, we have $(1-p)^{1/p} \geq e^{-\alpha}$ for large enough n. Therefore

$$E[Y] \geq n \left((1-p)^{1/p} \right)^{2t(n)r(n)/n} \geq n^{1-\alpha\varepsilon/2\ln 2} \ ,$$

since by hypothesis $t(n)r(n) \leq \frac{1}{4}\varepsilon n \log n$. Choosing $\alpha = 2\ln 2$ completes the proof. □

Now that we know that Y has high expectation, the main task is to prove that it does not fall far below its expectation too often. To this end we would like to consider an exposure martingale corresponding to Y that is obtained by revealing the values of Z_i one at a time. For $i = 0, 1, \ldots, n$, we define random variables $Y_i = Y_i(Z_1, \ldots, Z_i)$:

$$Y_i(z_1, \ldots, z_i) = E[Y | Z_1 = z_1, \ldots, Z_i = z_i] \ , \qquad \forall z_1, \ldots, z_i \in \{0,1\} \tag{3}$$

where the expectation is taken over $Z_{i+1}, \ldots, Z_n$. It is clear that $Y_{i-1}(z_1, \ldots, z_{i-1}) = (1-p)Y_i(z_1, \ldots, z_{i-1}, 0) + pY_i(z_1, \ldots, z_{i-1}, 1)$. Therefore, defining another set of random variables $D_i = D_i(Z_1, \ldots, Z_{i-1})$ by

$$D_i(z_1, \ldots, z_{i-1}) := Y_i(z_1, \ldots, z_{i-1}, 0) - Y_i(z_1, \ldots, z_{i-1}, 1) \ , \quad \forall z_1, \ldots, z_{i-1} \in \{0,1\} \tag{4}$$

gives

$$\begin{aligned} Y_i(z_1, \ldots, z_i) &= \begin{cases} Y_{i-1}(z_1, \ldots, z_{i-1}) + pD_i(z_1, \ldots, z_{i-1}) \ , & \text{if } Z_i = 0 \\ Y_{i-1}(z_1, \ldots, z_{i-1}) - (1-p)D_i(z_1, \ldots, z_{i-1}) \ , & \text{if } Z_i = 1 \end{cases} \\ &= Y_{i-1} + (p(1-Z_i) - (1-p)Z_i)D_i \\ &= Y_{i-1} + (p - Z_i)D_i \ , \end{aligned}$$

whence

$$Y = E[Y] + \sum_{i=1}^{n} (p - Z_i)D_i \ . \tag{5}$$

To bound the random variables D_i, note that for any fixed $z_1, \ldots, z_{i-1}$, the quantity D_i is a convex combination of the quantities

$$Y(\underbrace{z_1, \ldots, z_{i-1}}_{\text{fixed}}, 0, \underbrace{z_{i+1}, \ldots, z_n}_{\text{variable}}) - Y(z_1, \ldots, z_{i-1}, 1, z_{i+1}, \ldots, z_n) \tag{6}$$

where $(z_{i+1}, \ldots, z_n)$ varies over all tuples in $\{0,1\}^{n-i}$. From the definition of Y in (2), it is clear that each of the quantities (6) lies between 0 and d_i where

$$d_i := |\{F \in \mathcal{F} : i \in F\}| \ .$$

Therefore $0 \leq D_i \leq d_i$.

Lemma 3.6. *With $Z_1, \ldots, Z_n$ as above, let $c_1, \ldots, c_n$ be positive integers, let $C_i = C_i(Z_1, \ldots, Z_{i-1})$ be real functions satisfying $0 \leq C_i \leq c_i$ and let $\lambda > 0$ be an arbitrary real. Define $\Delta = \max_{i=1}^n c_i$ and $\delta = \max\{\lambda^2, \frac{1}{n}\sum_{i=1}^n c_i\}$. Then, if $pn \geq \Delta$ we have*

$$\Pr\left[\sum_{i=1}^{n} (p - Z_i) C_i \ < \ -\lambda \sqrt{pn\delta\Delta}\, \log n \right] \leq n^{-\lambda^2/6} \ .$$

Remark. The lemma is interesting because (1) we are summing *dependent* random variables and (2) it is a martingale inequality that does not follow from Azuma's inequality.

Proof. To simplify the proof, we assume that the C_i's are integer-valued; this assumption can easily be removed by discretizing with denominator n^2 and rescaling. For $1 \leq i \leq n$, $1 \leq j \leq d_i$, define random variables W_{ij} as follows

$$W_{ij} = \begin{cases} p - Z_i \ , & \text{if } j \leq C_i \\ 0 \ , & \text{if } j > C_i \ . \end{cases}$$

The key observation is that the nonzero W_{ij}'s for distinct i are independent, because the Z_i's are. Therefore, if we set $\ell = n\delta/\Delta$, the W_{ij}'s can be arranged in an $\ell \times \Delta$ matrix such that the nonzero random variables in each column are independent.[4] Fix a column and discard the zero entries; suppose m entries remain and sum to S. Standard Chernoff bounds (e.g. see [ASE00], Theorems A.11 and A.12) imply that for any $a > 0, b > 1$:

$$\Pr[S < -a] < \exp\left(-\frac{a^2}{2pm} + \frac{a^3}{2p^2m^2} \right) \ , \quad \text{and} \tag{7}$$

$$\Pr[S < -(b-1)pm] < \left(e^{b-1} b^{-b}\right)^{pm} \ . \tag{8}$$

Set $a = \lambda\sqrt{p\ell}\log n$ and $b = 1 + a/pm$. Suppose $a \leq \frac{2}{3}pm$. From (7) we immediately obtain

$$\Pr[S < -\lambda\sqrt{p\ell}\log n] < \exp(-\frac{1}{6}\lambda^2 \log^2 n) \leq \exp(-\frac{1}{6}\lambda^2 \log n) \ .$$

[4] We pad the matrix with extra zero entries, if necessary, to end up with the required shape of $\ell \times \Delta$.

Next suppose $a > \frac{2}{3}pm$. Since the real function $f(x) = (x-1)^{-1}x \ln x$ is increasing for $x \geq 1$, from (8) we get

$$\Pr[S < -\lambda\sqrt{p\ell}\log n] < \exp\left(-\lambda\sqrt{\frac{pn\delta}{\Delta}}\,(f(5/3)-1)\log n\right) < \exp(-\frac{1}{6}\lambda^2\log n) \ ,$$

where the last inequality from the facts that $pn \geq \Delta$ and that $\delta \geq \lambda^2$. Thus, in each case we have $\Pr[S < -\lambda\sqrt{p\ell}\log n] < n^{-\lambda^2/6}$.

To finish the proof we simply note that $\sum_{i=1}^{n}(p - Z_i)C_i$ is the sum of Δ such random variables S. □

We are now ready to prove our technical lemma.

Proof. **(of Lemma 3.3)** Let us apply Lemma 3.6 with $c_i = d_i$ and $C_i = D_i$. This choice of parameters gives $\Delta \leq s(n)$ and $\delta \leq \bar{s}(n)$. Because of the way we defined Y in (2), increasing p can only increase the quantity $\Pr[Y < \alpha]$, for any α; thus we may safely assume than $pn \geq \Delta$. Recalling that $p = 2t(n)/n$ we get

$$\Pr\left[Y < E[Y] - \lambda\sqrt{2t(n)s(n)\bar{s}(n)}\right] < n^{-\lambda^2/6} \ .$$

Recall that by hypothesis $t(n)s(n)\bar{s}(n) \leq n^{2-3\varepsilon}$. Using Lemmas 3.4 and 3.5 we then get

$$\Pr\left[X < n^{1-\varepsilon} - \lambda\sqrt{2n^{2-3\varepsilon}}\right] < 2n^{-\lambda^2/6} \ .$$

Setting $\lambda > \sqrt{12}$ yields $\Pr[X < \frac{1}{2}n^{1-\varepsilon}] < n^{-2}$ as desired, when n is large enough. □

4 Proof of the Packing Lemma

We now return to proving our improved packing lemma (Lemma 2.8). Recall that from the hypotheses we already have the following degree conditions on the bipartite graphs G and H we wish to pack:

$$\delta(G) \leq q \ , \quad \delta(H) \leq q \ , \quad \Delta_R(G) \leq q^2 \ , \quad \Delta_L(H) \leq q^2 \ , \tag{9}$$

where we have set

$$q = \left(\frac{\varepsilon}{16}n\log n\right)^{1/3} \ , \tag{10}$$

where ε is a small constant to be fixed later. We shall assume throughout this section that n is large enough.

Definition 4.1. *For a subset W of the vertex set of a graph and integer $k \leq |W|$, let $\mathcal{N}(W)$ denote the neighbourhood of W. Let $\mathrm{top}(W,k)$ and $\mathrm{bot}(W,k)$ denote the subsets of W consisting of, respectively, the k highest and k lowest degree vertices in W. For a vertex x, let $\mathcal{N}(x)$ be defined as $\mathcal{N}(\{x\})$.*

Following Hajnal [H91], our first step will be to modify G and H suitably so that even stronger degree conditions hold. Let $k = \min\{n/2, n/4\delta(H)\}$. From the hypotheses, we know that $V_R(G)$ has at least $n/2$ isolated vertices; let V_1 be a set of size $n/2$ of these. Let $V_0 = \text{top}(V_L(G), k)$, $V_2 = \text{bot}(V_L(H), k)$ and $V_3 = \mathcal{N}(V_2) \cup \text{top}(V_R(H), \frac{n}{2} - |\mathcal{N}(V_2)|)$. Let us define graphs G' and H' as follows:

$$G' = G - (V_0 \cup V_1) \ ; \qquad H' = H - (V_2 \cup V_3) \ .$$

It follows from the construction above that if G' and H' can be packed then so can G and H. This is because having packed G' and H' we may *arbitrarily* identify the vertices in V_0 with those in V_2 and the vertices in V_1 with those in V_3. Now, to show that G' and H' can be packed, we shall need the degree conditions guaranteed by the following lemma.

Lemma 4.2. *G' and H' are $(n - k, n/2)$-bipartite graphs with the following properties:*

$$\delta_L(G') \le q \ , \qquad \delta_R(G') \le q \ , \qquad \delta_L(H') \le q \ , \qquad \delta_R(H') \le q \ ,$$

$$\Delta_L(G') \le 4q^2 \ , \qquad \Delta_R(G') \le q^2 \ ,$$

$$\Delta_L(H') \le q^2 \ , \qquad \Delta_R(H') \le 4q \ .$$

Proof. The first four inequalities are obvious from (9), as are the bounds on $\Delta_R(G')$ and $\Delta_L(H')$. By construction $|\mathcal{N}(V_2)| \le \sum_{v \in V_2} \deg_H(v) \le n/4$; therefore V_3 contains at least $n/4$ of the highest degree vertices in $V_R(H)$. Since these vertices are removed to obtain H' we have $\Delta_R(H') \le 4\delta(H) \le 4q$. Similarly, we have $\Delta_L(G') \le 4\delta(H)\delta(G) \le 4q^2$. □

We prove that G' and H' can be packed using the probabilistic method: let $\varphi_L : V_L(G') \to V_L(H')$ be a random bijection; we shall show that with positive probability there exists a bijection $\varphi_R : V_R(G') \to V_R(H')$ such that (φ_L, φ_R) is a packing. Let $\Gamma = \Gamma(\varphi_L)$ be a bipartite graph on vertex set $(V_R(G), V_R(H))$ defined as follows: for $x \in V_R(G'), y \in V_R(H')$ we have $(x, y) \in E(\Gamma)$ iff $\varphi_L(\mathcal{N}(x)) \cap \mathcal{N}(y) = \emptyset$. It is clear that the required bijection φ_R exists iff the graph Γ has a perfect matching. Our task now is to show that the (random) bipartite graph Γ has a perfect matching with positive probability. The most straightforward way of doing this is to obtain lower bounds on the degrees of vertices in Γ and then apply König's Theorem.

The next two lemmas establish such lower bounds. It is important to note that unlike [H91] we exploit the asymmetry between G' and H' in a crucial way; the degree lower bound for H' is proved along lines similar to [H91], whereas for G' we need the power of our Technical Lemma.

Lemma 4.3. *With probability greater than $\frac{1}{2}$, for every vertex $y \in V_R(H')$ we have $\deg_\Gamma(y) \ge \frac{n}{2} - 8q^2$.*

Proof. Let $y \in V_R(H')$ be arbitrary and let $S(y) = \varphi_L^{-1}(\mathcal{N}(y))$. Then $S(y)$ is a random subset of $V_L(G')$ of size at most $\Delta_R(H') = 4q$. Since $\delta(G') \le q$, we have

$$E\left[\sum_{v \in S(y)} \deg_{G'}(v)\right] \le 4q^2 \ .$$

This bound on the expectation implies a high probability result proven through Chernoff bounds in exactly the same manner as Lemma 5.4 in [H91]; we need to have $\Delta_L(G') = O(n/\log n)$, but this is indeed the case by Lemma 4.2 and our choice of q in (10). Therefore, we get

$$\Pr\left[\sum_{v \in S(y)} \deg_{G'}(v) > 8q^2\right] < \frac{1}{2n} \ .$$

Thus $\Pr[\,|\mathcal{N}(S(y))| > 8q^2] < \frac{1}{2n}$ and so $\Pr[\deg_\Gamma(y) \ge \frac{n}{2} - 8q^2] < \frac{1}{2n}$. Since $|V_R(H')| \le n$, the lemma follows. □

Lemma 4.4. *With probability greater than* $\frac{1}{2}$*, for every vertex* $x \in V_R(G')$ *we have* $\deg_\Gamma(x) \ge 8q^2$.

Proof. Fix $\varepsilon = \frac{1}{10}$. Let $x \in V_R(G')$ be arbitrary and let $T(x) = \varphi_L(\mathcal{N}(x))$. Consider the set system $\mathcal{H} = \{\mathcal{N}(y) : y \in V_R(H')\}$ on ground set $V_L(H')$. By Lemma 4.2 we see that $\mathcal{H}$ is $(\frac{n}{2}, 4q, q^2, q)$-favourable and by (10) we have $4q \cdot q^2 = \frac{1}{4}\varepsilon n \log n$. Now $T(x)$ is a random subset of $V_L(H')$ of size $|T(x)| \le \Delta_R(G') \le q^2$ and $q^2 \cdot q^2 \cdot q = q^5 \le n^{2-3\varepsilon}$. Therefore, $\mathcal{H}$ and $T(x)$ satisfy the hypotheses of the Technical Lemma 3.3.

Applying the Technical Lemma, we see that the number of sets in $\mathcal{H}$ that $T(x)$ is disjoint from falls below $\frac{1}{2}n^{1-\varepsilon}$ with probability at most $\frac{1}{n^2} < \frac{1}{2n}$. In other words $\Pr[\deg_\Gamma(x) < \frac{1}{2}n^{1-\varepsilon}] < \frac{1}{2n}$. Noting that $\frac{1}{2}n^{1-\varepsilon} \ge 8q^2$ gives us the desired result. □

We now have all the pieces we need to prove the packing lemma.

Proof. **(Of the Packing Lemma)** From Lemma 4.3 and Lemma 4.4 we see that if the bijection φ_L is chosen at random, then with positive probability the following event occurs:

$$\forall x \in V_R(G') \quad \deg_\Gamma(x) \ge \frac{n}{2} - 8q^2 \ , \quad \text{and} \quad \forall y \in V_R(H') \quad \deg_\Gamma(y) \ge 8q^2 \ .$$

Since Γ is an $(n/2, n/2)$-bipartite graph, by König's Theorem, this event is a sufficient condition for Γ to have a perfect matching. Therefore, there exists a bijection φ_L such that Γ has a perfect matching; thus there exists a packing of G' and H'. By the discussion preceding Lemma 4.2 we see that G and H can be packed. □

5 Concluding Remarks

The complexity of graph properties has been studied for a quarter of a century now. Yet the most basic conjecture, namely an $\Omega(n^2)$ randomized decision tree complexity lower bound for monotone properties, remains open to this day. As mentioned before, the number of results leading towards a settlement of this conjecture have been very few.

Nine years have passed since the best previously known lower bound was established. We believe that this makes our result, a slight improvement of the bound, significant for injecting new life into this problem.

To improve the lower bound further it appears necessary to break out of Hajnal's framework. Our Technical Lemma is not constrained by this framework — instead, Hajnal's framework constrains the parameters we are forced to apply the lemma with — and we hope that it will be useful in further work on this problem.

Acknowledgments. We are grateful to Professor Andrew Yao for introducing us to this fascinating problem. We would like to thank Professor Bernard Chazelle for several important comments and suggestions.

References

[ASE00] Alon, N., Spencer, J. H. *The probabilistic method, Second Edition*, Wiley-Interscience Series, 2000, Appendix A.

[B78] Bollobás, B. *Extremal Graph Theory*, Academic Press, 1978, Chapter 8.

[G92] Gröger, H. D. *On the randomized complexity of monotone graph properties*, Acta Cybernetica, **10** (1992), 119–127.

[H91] Hajnal, P. *An* $\Omega(n^{\frac{4}{3}})$ *lower bound on the randomized complexity of graph properties*, Combinatorica, **11** (1991), 131–143.

[HS92] Hajnal, P., Szegedy, M. *On packing bipartite graphs*, Combinatorica, **12** (1992), 295–301.

[K88] King, V. *Lower bounds on the complexity of graph properties*, Proc. 20th ACM STOC (1988), 468–476.

[RV76] Rivest, R.L., Vuillemin, J. *On recognizing graph properties from adjacency matrices*, Theoret. Comput. Sci., **3** (1976), 371–384.

[Y87] Yao, A.C. *Lower bounds to randomized algorithms for graph properties*, Proc. 28th IEEE FOCS (1987), 393–400.

New Imperfect Random Source with Applications to Coin-Flipping

Yevgeniy Dodis

Department of Computer Science, New York University, 251 Mercer St, New York, NY 10012, USA. dodis@cs.nyu.edu

Abstract. We introduce a new *imperfect random source* that realistically generalizes the SV-source of Sántha and Vazirani [SV86] and the bit-fixing source of Lichtenstein, Linial and Saks [LLS89]. Our source is expected to generate a known sequence of (possibly dependent) random variables (for example, a stream of unbiased random bits). However, the realizations/observations of these variables could be imperfect in the following two ways: (1) inevitably, *each* of the observations could be *slightly* biased (due to noise, small measurements errors, imperfections of the source, etc.), which is characterized by the "statistical noise" parameter $\delta \in [0, \frac{1}{2}]$, and (2) *few* of the observations could be *completely* incorrect (due to very poor measurement, improper setup, unlikely but certain internal correlations, etc.), which is characterized by the "number of errors" parameter $b \geq 0$. While the SV-source considered only scenario (1), and the bit-fixing source — only scenario (2), we believe that our combined source is more realistic in modeling the problem of extracting quasi-random bits from physical sources. Unfortunately, we show that dealing with the *combination* of scenarios (1) and (2) is dramatically more difficult (at least from the point of randomness extraction) than dealing with each scenario individually. For example, if $b\delta = \omega(1)$, the adversary controlling our source can force the outcome of any bit extraction procedure to a constant with probability $1 - o(1)$, irrespective of the random variables, their correlation and the number of observations.
We also apply our source to the question of producing n-player collective coin-flipping protocols secure against *adaptive* adversaries. While the optimal non-adaptive adversarial threshold for such protocols is known to be $n/2$ [BN00], the optimal adaptive threshold is *conjectured* by Ben-Or and Linial [BL90] to be only $O(\sqrt{n})$. We give some evidence towards this conjecture by showing that there exists no *black-box transformation* from a *non-adaptively* secure coin-flipping protocol (with arbitrary conceivable parameters) resulting in an *adaptively* secure protocol tolerating $\omega(\sqrt{n})$ faulty players.

1 Introduction

Abstract Problem. Consider the following general problem. A sequence of dependent random variables $X_1, X_2, \ldots$ is generated (such a sequence is called a *stochastic process*). Ideally, each X_i has a known "ideal" conditional distribution, based on the outcomes of $X_1, \ldots, X_{i-1}$ (X_i's being independent is a special

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 297–309, 2001.

case). However, the "real" distributions with which the X_i's are generated could be slightly different from the expected "ideal" distribution. Moreover, in the applications we discuss later, it is typically the case that the exact nature of these "imperfections" is almost impossible to estimate or predict *exactly*. Therefore, we model them as if being caused by an *adversary* $\mathcal{A}$. On the optimistic side, we usually expect the "real" and the "ideal" stochastic process to be somewhat "close". In other words, there are some natural restrictions on the way $\mathcal{A}$ can influence our stochastic process. The general abstract problem is to determine how much "damage" can $\mathcal{A}$ cause subject to the restrictions put on it.

Let us now be more specific. Let N be the length of our stochastic process $\mathcal{P}$, and $D_i = D_i(x_1 \dots x_{i-1})$ be the ideal conditional distribution of X_i given $X_1 \dots X_{i-1} = x_1 \dots x_{i-1}$. Now, we will study the effects of two natural imperfections: inevitable (but small!) statistical deviation of each X_i from D_i, and rare (but complete!) errors in the process. More precisely, our adversary $\mathcal{A}$ knows $\mathcal{P}$ and is given the "noise" parameter $\delta \in [0, \frac{1}{2}]$ and the "error" parameter $b \geq 0$. Then, for any $i = 1 \dots N$ and given $x_1 \dots x_{i-1}$, $\mathcal{A}$ can influence the ideal sample of X_i using one of the following rules:

(A) Fix X_i to any constant in the support of D_i. This rule is called an *intervention* and can be used at most b times.
(B) Sample X_i from any distribution D'_i (on the same support set) of statistical distance[1] at most δ from D_i.

We notice that one of the most interesting ideal stochastic processes $\mathcal{P}$ is a sequence on N independent coin flips. In this important example, $\mathcal{A}$ observes $x_1 \dots x_{i-1} \in \{0,1\}^{i-1}$, and can affect the next coin x_i in the following two ways:

(A) *Fix* x_i to 0 or 1. Such an intervention can be used at most b times.
(B) *Bias* x_i by any value $\leq \delta$, i.e. set $\Pr(x_i = 1)$ anywhere inside $[\frac{1}{2} - \delta, \frac{1}{2} + \delta]$.[2]

We remark that in most of our applications, δ, b and N will be functions of some other implicit parameter (clear from the context). In such cases we will use asymptotic notation in this implicit parameter (i.e., $O(\cdot), \Omega(\cdot), o(\cdot), \omega(\cdot)$).

Motivation. We will show that the abstract setting described above turns out to be very relevant in at least the following three areas: (1) imperfect random sources; (2) discrete control processes; (3) black-box transformations from statically to adaptively secure distributed protocols. While each of the above applications will later deserve a separate section, we give a brief introduction now.

Imperfect Random Sources. A convenient abstraction in the design and analysis of various randomized algorithms is that the algorithm in question is given a

[1] Recall that the *statistical distance* between random variables Z and W over a domain R is $\|Z - W\| = \frac{1}{2} \cdot \sum_{\alpha \in R} |\Pr(Z = \alpha) - \Pr(W = \alpha)|$. The same notation stands for the distributions generating Z and W.
[2] Recall, a *bias* of a bit c is indeed defined to be $|\Pr(c = 1) - \frac{1}{2}|$.

stream of completely unbiased and independent random bits. In implementations, this stream has to be generated by some physical source of randomness. And obviously, such a physical source is unlikely to be "perfect": the realizations/observations of the randomness that it produces are bound to deviate from the "ideal" expected distribution (which actually need not be a sequence of unbiased random bits, even though the latter in an important special case). In particular, the following two "imperfections" are extremely natural: (1) inevitably, *each* of the observations could be *slightly* biased (due to noise, small measurements errors, etc.), and (2) *few* of the observations could be *completely* incorrect (due to poor measurement, improper setup, unlikely but certain internal correlations, etc.). Our abstract problem perfectly models the situation. Inevitable small noise is modeled by the "statistical noise parameter" $\delta \in [0, \frac{1}{2}]$ and the ability of the adversary to apply rule (B) above. Few total errors are modeled by the "number of errors" parameter $b \geq 0$ and the ability of the adversary to apply rule (A) a limited (at most b) number of times. While each of the imperfections alone seems to be insufficient to model a typical physical source, their combination seems to be much more realistic.

We also remark that the main question we address when looking at our problem from this angle is that of *randomness extraction*: can we apply some *deterministic* function to the observed output of our source so as to obtain nearly perfect randomness (even a single almost random bit!), despite the malicious behavior of the adversary?

Discrete Control Processes. This application is very similar to the above, except it looks at the problem from a different angle. Namely, given some random process $\mathcal{P}$, the question asked is how much "influence" over $\mathcal{P}$ is needed so as to force some desired event $\mathcal{E}$ (a function of $\mathcal{P}$'s output) to happen. In this sense, our adversary $\mathcal{A}$ can be seen as a *controller* trying to minimize the usage of its "resources" and still force $\mathcal{E}$ to happen. Again, our problem models the situation quite naturally. Rule (A), where $\mathcal{A}$ can completely fix the outcome of X_i, is the "expensive" resource that $\mathcal{A}$ tries to minimize. It also explains while we call rule (A) an "intervention". On the other hand, rule (B), where $\mathcal{A}$ can just slightly (or not at all if $\delta = 0$) affect each X_i, can be viewed as something that takes "no effort" for $\mathcal{A}$ to perform. A good analogy in this scenario could be that rule (A) corresponds to a "sharp turn" or "changing highway", while rule (B) to a "casual streering" or "changing lane".

While the "rules of the game" are the same as for imperfect random sources, the main difference is in the question addressed: given the desired event $\mathcal{E}$, either tell the smallest expected number of $\mathcal{A}$'s interventions so as to *guarantee* $\mathcal{E}$, or tell smallest probability of $\mathcal{A}$'s failure for a fixed number of interventions.

Black-box Transformations. Assume we have a distributed protocol for n players to flip a coin or, more generally, to sample some distribution D. As usual, we can assume that some number of players (say, b) is malicious, and is controlled by a central adversary $\mathcal{A}$. As a result, the malicious players can somewhat bias the resulting distribution D', and we try to design protocols where such bias

is guaranteed to be small. A crucial distinction made in the design of such protocols, is whether the adversary $\mathcal{A}$ is *static* or *adaptive*. In the former case $\mathcal{A}$ has to decide which b players to "corrupt" before the protocol starts, while in the latter case it can make these decisions dynamically in the course of the execution. It turns out that it is significantly easier to design statically secure protocols than their dynamic counterparts. The question we address is whether and when it is possible to achieve adaptive security "for free". More specifically, can we transform some good (family of) statically secure protocol(s) Π so as to obtain a reasonable adaptively secure protocol Φ (for the same or related task)? Moreover, the proof of Φ's adaptive security should only depend on the knowledge that Π is statically secure and not on any other specifics about Π (precisely, we will only assume that static $\mathcal{A}$ corrupting b players can bias the outcome of Π by at most $\delta \in [0, \frac{1}{2}]$).

We formalize this using the notion of a *black-box transformation*. Namely, we will sequentially run (various protocols in) Π many (say, N) times against an adaptive adversary $\mathcal{A}$ who can dynamically corrupt up to b players. These runs were expected to produce outputs $X_1 \dots X_N$. Of course, since we run a static protocol against an adaptive adversary, some of the X_i's might be very biased. However, $\mathcal{A}$ can corrupt at most b players! Thus, at least $(N - b)$ of the subprotocols were effectively run against a *static* adversary, and therefore produced outputs with the bias at most δ. But then, even if the other b runs produced X_i's which were completely fixed by $\mathcal{A}$ (which actually *happens*, say, in current static coin-flipping protocols), we exactly get our abstract problem!

The question addressed here is for which setting of parameters such black-box transformations are possible.

Our Results. We will study our abstract problem and show that the adversary is quite powerful for essentially any non-trivial setting of parameters. In particular, applying our results to the three motivating applications above, we show the following. If $b\delta = \omega(1)$ and *independent of the number of samples* N: (1) no "non-trivial" distribution Y (e.g., a random bit) can be sampled from our imperfect source; (2) any "non-constant" event $\mathcal{E}$ can be forced with probability $1 - o(1)$ (alternatively, $O(1/\delta)$ expected interventions suffices to *force* $\mathcal{E}$); (3) no black-box transformation can result in a "non-trivial" adaptive sampling protocol. The latter result is extended to show that no black-box transformation from *any* statically secure n-player coin-flipping protocol can result in adaptively secure coin-flipping protocol tolerating $\omega(\sqrt{n})$ corruptions, giving support to a conjecture of Ben-Or and Linial [BL90].

Organization. While all our results hold for general stochastic processes, the special case when $\mathcal{P}$ is a stream of unbiased bits will turn out to be quite representative of the general situation, but will substantially simplify the presentation. Therefore, we will mainly restrict our attention to the stream of unbiased bits.

In Section 2 we talk about the "imperfect random source view" on our problem. In particular, we completely characterize the (im)possibility of bit extraction from our *bias-control limited* (BCL) source. Next, in Section 3 view our

source as a discrete control process. We derive tight bounds on how "influential" our adversary (or the controller) is in this regard. In Section 4 we have our main application to collective coin-flipping: impossibility of black-box transformations from statically to good adaptively secure protocols.

2 Imperfect Random Sources

Prior Work. Much work has been done on imperfect random sources. Due to space constraints, we only survey the relevant to us history of *streaming* sources. Like the ideal source, such sources produce a stream of bits (recall, we will talk only about bits for simplicity) incrementally over time, but these bits are not necessarily unbiased or independent. Perhaps the first streaming source goes all the way back to von Newman [vN51] who showed how to extract perfect random bits from a sequence of *independent* coin tosses of the *same* biased coin (of unknown bias). Elias [E72] showed how to improve this result and extract perfect random bits at the optimal rate. Blum [B86] relaxed the independence requirement on the source by considering streaming sources generated by finite-state Markov chains (of *unknown* structure).

The next important development was made by Sántha and Vazirani [SV86] who considered a more general streaming source, called a *semi-random source* (or an *SV-source*). In this source each subsequent bit can be arbitrarily correlated with *all* the previous bits, as long as it has some uncertainty. More specifically, the source is specified by the "noise" parameter $\delta \in [0, \frac{1}{2}]$, and can produce any sequence $x_1, x_2, \ldots$ as long as $\Pr(x_i = 1 \mid x_1 \ldots x_{i-1}) \in [\frac{1}{2} - \delta, \frac{1}{2} + \delta]$. This source tries to model the fact that physical sources can never produce *completely* perfect bits (anyway, our observation of such sources is bound to introduce some noise). Alternatively, the stream of bits could be produced by a distributive coin-flipping protocol [BL90], where few malicious players can slightly bias each of the bits.

In a parallel development, Lichtinstein, Linial and Saks [LLS89] considered another streaming source, called the (adaptive) *bit-fixing* source. In this source (characterized by the "number of errors" parameter b) each next bit, depending on the previous bits, can be either perfectly random (which is one of the main limitations of this source) or completely fixed to 0 or 1. The only constraint is that at most b of the bits are fixed. This source tries to model the situation that some of the bits generated by a physical source could be *determined* from the previous bits, even though we assume that this does not happen very frequently (at most b times). Alternatively, it relates to the study of "discrete control processes" that we mentioned earlier, as well as to the problem of adaptive coin-flipping where each player sends at most one bit (see Section 4).

Our Source. As we see already, our new streaming source examines the implications of having *both* the problems of "constant small noise" and "rare total errors", naturally generalizing random sources of [SV86, LLS89]. Interestingly, we will show that having both imperfections together is significantly more difficult to deal with than having any of them individually, but first we need some notation. We call our source (given by δ, b, N and a particular adversary $\mathcal{A}$ obeying

rules (A) and (B)) *Bias-Control Limited*, or simply (δ, b, N)-BCL source. Notice, $b = 0$ (or applying only rule (B)) corresponds to the SV-source, $\delta = 0$ (or applying only rule (A)) yields the bit-fixing source, while $b = \delta = 0$ gives the perfect randomness. Now we can quantitatively measure the "goodness" of our source for the problem of bit extraction.

Definition 1. *Let $\mathcal{A}$ be some (δ, b, N)-BCL source, and $f : \{0,1\}^N \to \{0,1\}$ be a bit extraction function. Let*

- *$q(\delta, b, N, f, \mathcal{A})$ be the bias of $f(x)$, where $x = x_1 \dots x_N$ was produced by $\mathcal{A}$.*
- *$q(\delta, b, N, f) = \max_{\mathcal{A}} q(\delta, b, N, f, \mathcal{A})$ (taken over all (δ, b, N)-BCL sources $\mathcal{A}$).*
- *$q(\delta, b, N) = \min_f q(\delta, b, N, f)$ (taken over all $f : \{0,1\}^N \to \{0,1\}$).*

Thus, $q(\delta, b, N)$ is the smallest bias of a coin that can be extracted from any (δ, b, N)-BCL source.

We will say that one can extract an *almost* perfect bit from a (δ, b, N)-BCL source, if $q(\delta, b, N) = o(1)$, and a *slightly* random bit if $q(\delta, b, N) \le \frac{1}{2} - \Omega(1)$. We will now survey the known results about the SV-source and the bit-fixing source, and then parallel them with our results.

Extraction from the Bit-Fixing Source. Recall, the bit-fixing source of [LLS89] corresponds to having b interventions and $\delta = 0$. Notice, that if we let f to be the majority function, we can tolerate $b = O(\sqrt{N})$ since any $c\sqrt{N}$ bits (for small enough constant c) do not influence the resulting (almost random) value of majority with probability $1 - o(1)$. Remarkably enough, Lichtinstein, Linial and Saks [LLS89] actually showed that this is the best bit extraction possible.

Theorem 1 ([LLS89]). *Majority is the best bit extraction function for the bit-fixing source: $q(0, c_1\sqrt{N}, N) = o(1)$, while $q(0, c_2\sqrt{N}, N) = \frac{1}{2} - o(1)$ ($c_1 < c_2$).*

As a side note, a *random* function $f : \{0,1\}^N \to \{0,1\}$ is a terrible bit extraction function for the bit-fixing source even for $b = \omega(1)$. Indeed, with high probability the first $(N - b)$ bits do not fix f, so $\mathcal{A}$ can use the last b interventions to fix f. Another terrible function (even for $b = 1$) is any parity function: it can be fixed by fixing the last emitted bit.

Extraction from the SV-source. Recall, the SV-source [SV86] corresponds to having $b = 0$, and where $\Pr(x_i = 1 \mid x_1 \dots x_{i-1}) \in [\frac{1}{2} - \delta, \frac{1}{2} + \delta]$. On a negative side, Sántha and Vazirani showed that one cannot extract a bit whose bias is less than δ. In other words, many samples (i.e., large N) from the source do not help: outputting x_1 is as good as we can get! Notationally,

Theorem 2 ([SV86]). *$q(\delta, 0, N) = \delta$. Thus, one can extract an almost perfect bit iff $\delta = o(1)$, and a slightly random bit iff $\delta = \frac{1}{2} - \Omega(1)$,*

Clearly, there are many (optimal) functions that extract a δ-biased coin from any SV-source: for example, any parity function will do. In fact, Boppanna and Narayanan [BN96] (extending the ideas of [AR89]) show that a vast majority of boolean functions from N bits extract a slightly random bit (provided, of course,

$\delta = \frac{1}{2} - \Omega(1)$). Unfortunately, majority is not one of these functions (unless $\delta \ll 1/\sqrt{N}$, which will turn out to be important soon). Indeed, if our source always sets the 1-probability of the next bit to be $\frac{1}{2} + \delta$, the resulting bit will be 1 with probability $1 - o(1)$. In fact, Alon and Rabin [AR89] showed that *majority is the worst* bit extracting function. Namely, $q(\delta, 0, N, majority) \geq q(\delta, 0, N, f)$, for any f.

Extraction from Our Source. Looking at the extreme cases of our source ($\delta = 0$ and $b = 0$), we notice that somewhat reasonable bit extraction (at least of slightly random bits) is possible for both of them. However, the extraction functions are diametrically opposite. For the bit-fixing source the best function was the majority, and a random function (or any parity function) was terrible, while for the SV-source a random function was good (and any parity function is optimal), and the majority was the worst. Hence, the best bit extractor becomes the worst and vice versa! One may wonder if some extraction function can work reasonably well for both of these extreme cases, and hopefully provide a good extraction for our combined source as well. Unfortunately, we show that such a magic function does not exist for (any "interesting" setting of) our combined source. The following theorem follows from Theorem 4 in Section 3:

Theorem 3. *If $b\delta = \omega(1)$, then it is impossible to extract a slightly random bit from a (δ, b, N)-BCL source, irrespective of the value of N! More precisely,*

$$q(\delta, b, N) \geq \frac{1}{2} - \frac{2}{(1+2\delta)^b} = \frac{1}{2} - \frac{1}{2^{\Omega(\delta b)-1}} \tag{1}$$

In particular, while for $\delta = 0$ we could tolerate $b = O(\sqrt{N})$, and for $b = 0$ could deal with $\delta < \frac{1}{2} - \Omega(1)$, now we cannot tolerate $b \to \infty$ for any (constant) $\delta > 0$, no matter how large N is. Notice also that the worst-case bias of any extracted coin *exponentially* approaches to $\frac{1}{2}$ as b grows.

Tightness. First, given b and δ, let us see under which conditions on N the majority on N bits will be a good bit extraction for the (δ, b, N)-BCL source? A moment look at the binomial distribution reveals that if $N \ll b^2$, b interventions allow the adversary to almost control the coin. On the other hand, if $N \gg 1/\delta^2$, then the δ-bias at every step again allows the adversary to almost control the coin. Hence, if $b^2 \gg 1/\delta^2$, i.e. $b\delta \gg 1$, then no N will make the majority "good". This is not surprising in light of Theorem 3, but the converse statement is more interesting. It is easy to show that if $b^2 \ll 1/\delta^2$, i.e. $b\delta \ll 1$, *any* N such that $b^2 \ll N \ll 1/\delta^2$ will result in the majority being a good extractor (in fact, $N \approx b/\delta$ is the best). But what if $N > 1/\delta^2$? Of course, the majority itself does not work then. However, we can still trivially extract an almost random bit by simply ignoring some (say, the first or the last) $N - O(1/\delta^2)$ bits and taking the majority of the remaining $O(1/\delta^2)$ bits! Collecting these ideas, we get

Lemma 1. *If $b\delta = O(1)$, $b = O(\sqrt{N})$ and $\delta = o(1)$, one can extract an almost random bit from a (δ, b, N)-BCL source: $q(\delta, b, N) = o(1)$. In particular, this can be done by taking majority of any $\min(N, O(1/\delta^2))$ bits of the source.*

Complete Picture. We also notice that Theorem 3 does not imply Theorems 1 and 2, which study the extreme cases of our source. However, by *combining* all three results with Lemma 1, we get a complete characterization of the of bit extraction picture from any (δ, b, N)-BCL source. Namely, the following list covers all the significant cases:

1. If $b = \Omega(\sqrt{N})$ or $\delta = \frac{1}{2} - o(1)$ or $b\delta = \omega(1)$, it is impossible to extract even a slightly random bit. These results follow from Theorem 1 (even for $\delta = 0$), Theorem 2 (even for $b = 0$) and Theorem 3 respectively.
2. If $\Omega(1) \le \delta \le \frac{1}{2} - \Omega(1)$ and $b = O(1)$, then one can extract a slightly but not almost random bit (the lower bound follows from Theorem 2).
3. If $b = O(\sqrt{N})$ and $b\delta = O(1)$ and $\delta = o(1)$, then one can extract an almost random bit from our source. This is exactly Lemma 1.

To have yet another insight on these results, we can let $\sigma \stackrel{\text{def}}{=} \max(\delta, O(1/\sqrt{N}))$ to be the "effective noise" of our source. In other words, if $\delta \ll 1/\sqrt{N}$, increasing δ to $1/\sqrt{N}$ does not change the behavior of the source much. Then we can restate our main result as follows: when $b\sigma = \omega(1)$, no good extraction is possible, and if $b\sigma = O(1)$, good extraction becomes possible.

Expected Number of Interventions to Fix the Outcome. We also study another bit extraction measure of our source: the *expected* number of interventions to *always* fix the extracted coin (to 0 or 1). Due to space limitations, we leave it to the final version, where we show that $O(1/\delta)$ expected interventions suffice irrespective of N. Combining with earlier results of [LLS89] for $\delta = 0$ (that $O(\sqrt{N})$ interventions suffice), we conclude that the right answer is $\Theta(\min(1/\delta, \sqrt{N})) = \Theta(1/\sigma)$.

Sampling General Distributions. We can look at the question of sampling general distributions, and not just a single coin-flip. Not surprisingly, since we could not even sample a slightly random bit from our source, the same will hold for other distributions. Namely, if $b\delta = \omega(1)$ and f ideally extracts a non-trivial $Y = f(X)$ (i.e., there is no y s.t. $\Pr(Y = y) = 1 - o(1)$) from our source, then $\mathcal{A}$ can influence X to X' such that $Y' = f(X')$ is *statistically far* from Y: $\|Y - Y'\| \ge \frac{1}{2} - o(1)$. Thus, no extraction is possible. We leave the details to the final version.

3 Discrete Control Processes

Alternative View of Our Source. So far we considered the task of our adversary $\mathcal{A}$ to be preventing good bit extraction. However, an equally (if not more) natural task for $\mathcal{A}$ would be to try to force some particular *event* $\mathcal{E}$, i.e. to force the string $x = x_1 \dots x_N$ to satisfy some particular property. To formalize this, let $\mathcal{E}$ be an event (or property) on $\{0,1\}^N$. Equivalently, $\mathcal{E}$ can be viewed as a boolean function $e : \{0,1\}^N \to \{0,1\}$, or as a *language* $L = e^{-1}(1) \subseteq \{0,1\}^N$, via "$\mathcal{E}$ happened $\iff$ $e(x) = 1$ $\iff$ $x \in L$". We define the *natural probability* p of $\mathcal{E}$ to be the probability that $\mathcal{E}$ happened for the *ideal* source (in our special

case, emitting N perfect unbiased bits), i.e. $p = |L|/2^N$, and say that $\mathcal{E}$ is *p-sparse*. Now we want to see if our adversary $\mathcal{A}$ has enough power to significantly influence the occurrence of $\mathcal{E}$ (i.e., to make $x \in L$). Two dual questions naturally come up for a given δ, N and $\mathcal{E}$ (with natural probability p):

1. For a given number of interventions b, what is the smallest probability of "failure" that $\mathcal{A}$ can achieve? In particular, under what conditions can it be arbitrarily close to 0? Can the answer(s) depend on p but not on other specifics of $\mathcal{E}$?
2. Assume we want to *guarantee* success ($\mathcal{E}$ always happens), by allowing possibly unbounded number of interventions. What is the smallest *expected* number of interventions needed? Can the bound depend on p but not on other specifics of $\mathcal{E}$?

We define two natural measures that allow us to study the quantities addressed in the above two questions. Since δ is never going to change in our discussion, we omit it from all the notation below.

Definition 2. *Define*

- $F(p, N, b) = \max_{\mathcal{E}} \min_{\mathcal{A}} \Pr(e(x) = 0)$*, taken over all p-sparse* $\mathcal{E}$*, and all* (δ, b, N)*-BCL* $\mathcal{A}$*.*
- $B(p, N) = \max_{\mathcal{E}} \min_{\mathcal{A}} \mathbf{E}[b]$*, taken over all p-sparse* $\mathcal{E}$ *and all N-bit sources* $\mathcal{A}$ *(with noise* δ*) necessarily producing* x *satisfying* $\mathcal{E}$*. Here* $\mathbf{E}[b]$ *stands for the expected number of interventions used by* $\mathcal{A}$ *(over the usage of rule (B)).*

Thus, $F(p, N, b)$ is the largest probability of $\mathcal{A}$'s failure over all p-sparse events, and $B(p, N)$ is the smallest expected number of interventions $\mathcal{A}$ needs to always force any p-sparse $\mathcal{E}$. Notice, both times we take the worst-case p-sparse $\mathcal{E}$.

Bounding the Probability of Failure. We start with a tight bound on $F(p, N, b)$.

Theorem 4.

$$F(p, N, b) \leq \frac{1}{p \cdot (1 + 2\delta)^b} = 2^{\log(1/p) - \Theta(\delta b)} \tag{2}$$

Thus, if $\delta b = \omega(\log(\frac{1}{p}))$*,* $\mathcal{A}$ *can force any p-sparse* $\mathcal{E}$ *with probability* $1 - o(1)$*.*

Several remarks are in place before the proof. First, N does not enter the equation above. Second, Theorem 4 immediately implies Theorem 3 (since for any extraction function f, either the event $f(x) = 0$ or the event $f(x) = 1$ has natural probability $p \geq 1/2$). Finally, the bound in Equation (2) is almost tight, at least in several significant cases. For example, for $p = 1/2$, Lemma 1 implies that $\mathcal{A}$ cannot almost certainly force 1 on the majority of $\min(N, 1/\delta^2)$ bits when $\delta b = O(1)$. On the other hand, if e is the function that is 1 on the first $p2^N$ values of x (in the lexicographic order), $\mathcal{A}$ has to intervene at least $\Omega(\log(1/p))$ times in order to force $e(x) = 1$ with probability more than $\frac{1}{2} + \delta$.

Proof. The statement is true for $\delta = 0$ or $b = 0$, since $F(\cdot,\cdot,\cdot) \le 1 \le 1/p$, so assume $\delta > 0$ and $b \ge 1$. Define $g(p,b) = \frac{1}{p(1+2\delta)^b}$. We need to show that $F(p,N,b) \le g(p,b)$ for any $N \ge 1$, $1 \le b \le N$ and $0 \le p \le 1$. We prove this by induction on N. For $N = 1$, $F(0,1,b) = 1 < \infty = g(0,b)$, and $F(p,1,b) = 0 \le g(p,b)$ for $p > 0$ (here we used $b \ge 1$). Now assume the claim is true for $(N-1)$.

Take any p-sparse $\mathcal{E}$ given by a function e. Let $e_0 : \{0,1\}^{N-1} \to \{0,1\}$ be the restriction of e when $x_1 = 0$. Similarly for e_1. This defines a p_0-sparse event $\mathcal{E}_0$ and a p_1-sparse event $\mathcal{E}_1$ satisfying $\frac{1}{2}(p_0 + p_1) = p$. Without loss of generality assume $p_0 \ge p \ge p_1$. Given such $\mathcal{E}$, our particular adversary $\mathcal{A}$ will consider two options and pick the best (using its unbounded computational resources): either use an intervention (which is legal since we assumed $b \ge 1$) and fix $x_1 = 0$, reducing the question to that of analyzing the p_0-sparse event $\mathcal{E}_0$ on $(N-1)$ variables and also reducing b by 1, or use rule (B) making the 0-probability of x_1 equal to $\frac{1}{2} + \delta$ and leaving the same b. We get the following recurrence:

$$F(p,N,b) \le \min[\ F(p_0, N-1, b-1),\ \left(\frac{1}{2} - \delta\right) \cdot F(p_1, N-1, b) + \left(\frac{1}{2} + \delta\right) \cdot F(p_0, N-1, b)\ ]$$

Let $p_0 = p(1+\beta)$ and $p_1 = p(1-\beta)$, where $0 \le \beta \le 1$ (since $p_0 \ge p \ge p_1$). Using our inductive assumption,

$$F(p,N,b) \le \min(\ g(p(1+\beta), b-1),\ \left(\frac{1}{2} - \delta\right) \cdot g(p(1-\beta), b) + \left(\frac{1}{2} + \delta\right) \cdot g(p(1+\beta), b)\ ) \stackrel{?}{\le} g(p,b)$$

Recalling the definition of g, it thus suffices to show that

$$\min\left(\frac{1}{p(1+\beta)(1+2\delta)^{b-1}}, \frac{\frac{1}{2}-\delta}{p(1-\beta)(1+2\delta)^b} + \frac{\frac{1}{2}+\delta}{p(1+\beta)(1+2\delta)^b}\right) \le \frac{1}{p(1+2\delta)^b}$$

$$\iff \qquad \min\left(\frac{1+2\delta}{1+\beta}\ ,\ \frac{\frac{1}{2}-\delta}{1-\beta} + \frac{\frac{1}{2}+\delta}{1+\beta}\right) \le 1$$

We show that the inequality above holds for *any* $\beta \in [0,1]$ (since the choice of β is outside of our control). We see that the expressions under the minimum are equal when $\beta = 2\delta$. The following two cases on β complete the proof.

- Case 1. Assume $\beta \ge 2\delta$. Then the minimum above is $\frac{1+2\delta}{1+\beta}$ and we need to show that $\frac{1+2\delta}{1+\beta} \le 1$, which is equivalent to $\beta \ge 2\delta$.
- Case 2. Assume $\beta \le 2\delta$. Then the minimum above is $\frac{\frac{1}{2}-\delta}{1-\beta} + \frac{\frac{1}{2}+\delta}{1+\beta}$ and we need to show that $\frac{\frac{1}{2}-\delta}{1-\beta} + \frac{\frac{1}{2}+\delta}{1+\beta} \le 1$, which is equivalent to $\beta \le 2\delta$.

Bounding Expected Number of Interventions. We also show a tight bound on $B(p,N)$. Namely, $B(p,N) = O(\frac{1}{\delta}\log(\frac{1}{p}))$ (in particular, this bound is independent on N). Due to space limitations, we leave the proof to the final version.

Generalizing to Any Stochastic Process. As we mentioned earlier, our results can be generalized to any stochastic process $\mathcal{P}$. Namely, the notion of natural probability p of $\mathcal{E}$ and the quantities $F_{\mathcal{P}}(p, N, b)$ and $B_{\mathcal{P}}(p, N)$ can now be defined w.r.t. to $\mathcal{P}$ completely analogously to Definition 2. While the proofs become significantly more involved, in the final version we show for *any* $\mathcal{P}$: $F_{\mathcal{P}}(p, N, b) \leq (1-\delta)^b/p = 2^{\log(1/p)-\Omega(\delta b)}$; $B_{\mathcal{P}}(p, N) \leq \log_{1-\delta} p = O(\frac{1}{\delta} \cdot \log(\frac{1}{p}))$.

4 Black-Box Transformations and Adaptive Coin-Flipping

The Setting. Collective Coin-Flipping, introduced by Ben-Or and Linial [BL90], is a problem where n (computationally unbounded) processors are trying to generate a random bit in a setting where only a *single broadcast channel* is available for communication. At most b out of n players can be controlled by a central adversary $\mathcal{A}$ (which is called b-bounded) who is trying to bias the resulting coin. Given a protocol Π, we let $\Delta_\Pi(b)$ be the largest bias achieved by a b-bounded adversary against Π. Π is said to be $b(n)$-*resilient* if Π produces a *slightly* random coin: $\Delta_\Pi(b(n)) \leq \frac{1}{2} - \Omega(1)$, where the constant is *independent* of n. Similarly, Π is said to be *strongly* $b(n)$-*resilient* if Π produces an *almost* random coin: $\Delta_\Pi(b(n)) = o(1)$. As we said in the introduction, it makes a crucial difference whether $\mathcal{A}$ is static (decides whom to corrupt before the protocol starts), or adaptive (decides whom to corrupt during the protocol).

Coin-Flipping with Static Adversaries. The optimal resilient threshold for static adversaries in $n/2$: any $n/2$ players can always fix the coin [S89,BN00], while there exist $(\frac{1}{2} - \varepsilon)$-resilient protocols (even constructive and efficient ones) for any $\varepsilon > 0$ [BN00,RZ98,F99]. We also point out a very simple dependence of the optimal bias $\Delta(b)$ (defined to be the smallest bias achieved by a coin-flipping protocol: $\min_\Pi \Delta_\Pi(b)$) on the number of players: $\Delta(b) = \Theta(b/n)$ [BL90,AN93]. Finally, we point out that *all the best statically secure coin-flipping protocols are not even* 1*-resilient against adaptive adversaries*. This is due to a historical feature that all such protocols first elect a single (hopefully, not faulty) representative player (called a *leader*), who then flips the final coin by itself. Corrupting such a leader at the end clearly controls the coin.

Coin-Flipping with Adaptive Adversaries. Adaptive adversaries were already considered by Ben-Or and Linial [BL90], who observed that the "majority" protocol (each player sends a random bit, and the final coin is their majority) achieves adaptive $\Theta(\sqrt{n})$-resilience. Surprisingly enough, this simple protocol is the *best known* adaptively secure coin-flipping protocol! In fact, Ben-Or and Linial [BL90] conjectured that this protocol to be optimal! This conjecture (call it (*)), if true, would imply that adaptive adversaries are much more powerful than static adversaries (where the threshold is $n/2$) for the problem of collective coin-flipping. Interestingly enough, the only result that in support of conjecture (*) comes from the bit-fixing source of [LLS89]. Namely, when each player sends only 1 bit in the entire protocol, the optimal behavior of the adversary is exactly

the same as in the bit-fixing source with b interventions! Since the majority was the best bit extraction function for the bit-fixing source, conjecture (*) is true in this case. This result is interesting since is already illustrates the power of adaptivity. Namely, in the static case one can achieve $\Omega(n/\log^2 n)$-resilience [AL93] when players send only 1 bit, even in one round. However, it supports the conjecture (*) much less than it seems to. Indeed, restricting each player to send at most 1 bit seems like a huge limitation. For example, it is very limiting *even for statically secure protocols* (i.e., no function can be more than $O(n/\log n)$-resilient by the result of [KKL89], and there are general $n/2$-resilient statistically secure protocols [BN00,RZ98,F99]).

To summarize, adaptively secure coin-flipping is much less understood than its static counter-part, there seems to be some indication that adaptive adversaries are much more powerful than static adversaries, but there is little formal evidence supporting this claim.

Black-Box Reductions. Due to space limitations, we leave the formal treatment to the final version, and instead provide informal (but informative) intuition of our approach, which we already sketched in the introduction. Namely, we want to sequentially run a static coin-flipping protocol Π for N times, and try to extract a good coin from the N outcomes $x_1 \dots x_N$. If $\delta = \Delta_\Pi(b)$, then we reduced the adaptive adversary $\mathcal{A}$ to a (δ, b, N)-BCL source: rule (A) corresponds to corrupting a player during one of the N sub-protocols, while rule (B) corresponds to not doing so and using the power of the static adversary. Notice, while b and δ are fixed (given n), we have the power to make N really huge, which seems to give us a considerable advantage. Unfortunately, the strong negative result of Theorem 3 shows that this advantage is, actually, an illusion. Namely, our results say that the possibility of bit extraction from our source depends on whether $b\delta = O(1)$ or $b\delta = \omega(1)$, i.e. a large number of repetitions N does *not* help.

Nevertheless, when is $b\delta = O(1)$? Notice that the best δ we could hope for (without looking at the specifics of Π), while definitely no *more* than $\Delta(b)$, can not be much *less* than $\Delta(b) = \Theta(b/n)$ as well. For example, at the very beginning $\mathcal{A}$ could corrupt $b/2$ players that gives $\delta \geq \Delta(b/2) = \Theta(\Delta(b))$, and still have $b/2$ arbitrary corruptions left. Hence, our "black-box" approach can work (and actually *can be made* to work) only if $b \cdot \Theta(b/n) = O(1)$, i.e. $b = O(\sqrt{n})$. Since such b can be trivially achieved by the majority protocol, we cannot achieve adaptive security (beyond what is known) "for free".

Discussion. We are not saying that black-box transformations are the most natural way to achieve adaptive security. However, the "breaking point" of our approach is exactly (believed to be optimal) $b = \Theta(\sqrt{n})$. The latter "coincidence" does give some further evidence to conjecture (*).

References

[AL93] M. Ajtai, N. Linial. The influence of large coalitions. *Combinatorica*, 13(2):129–145, 1993.

[AN93] N. Alon, M. Naor. Coin-flipping games immune against linear-sized coalitions. *SIAM J. Comput.*, 22(2):403-417, 1993.

[AR89] N. Alon, M. Rabin. Biased Coins and Randomized Algorithms. *Advances in Computing Research*, 5:499-507, 1989.

[BL90] M. Ben-Or, N. Linial. Collective Coin-Flipping. In *Randomness and Computation*, pp. 91-115, Academic Press, New York, 1990.

[B86] M. Blum. Independent unbiased coin-flipsfrom a correclated biased source — a finite state Markov chain. *Combinatorica*, 6(2):97–108, 1986.

[BN96] R. Boppana, B. Narayanan. The Biased Coin Problem. *SIAM J. Discrete Math.*, 9(1)29–36, 1996.

[BN00] R. Boppana, B. Narayanan. Perfect-information Leader Election with Optimal Resilience. *SIAM J. Comput.*, 29(4):1304-1320, 2000.

[E72] P. Elias. The Efficient Construction of an Unbiased Random Sequence. *Ann. Math. Stat.*, 43(3):865–870, 1972.

[F99] U. Feige. Noncryptographic Selection Protocols. In *Proc. of* 40*th FOCS*, pp. 142–152, 1999.

[KKL89] J. Kahn, G. Kalai, N. Linial. The Influence of Variables on Boolean Functions. In *Proc. of* 30*th FOCS*, pp. 68–80, 1989.

[LLS89] D. Lichtenstein, N. Linial, M. Saks. Some Extremal Problems Arising from Discrete Control Processes. *Combinatorica*, 9:269–287, 1989.

[RZ98] A. Russell, D. Zuckerman. Perfect information leader election in $\log^* n + O(1)$ rounds. In *Proc. of* 39*th FOCS*, pp. 576–583, 1998.

[S89] M. Saks. A robust noncryptographic protocol for collective coin flipping. *SIAM J. Discrete Math.*, 2(2):240–244, 1989.

[SV86] M. Sántha, U. Vazirani. Generating Quasi-Random Sequences from Semi-Random Sources. *J. of Computer and System Sciences*, 33(1):75–87, 1986.

[vN51] J. von Newman. Various techniques used in connection with random digits. In *National Bureau of Standards, Applied Math. Series*, 12:36–38, 1951.

Recognizing More Unsatisfiable Random 3-SAT Instances Efficiently

Joel Friedman[1] and Andreas Goerdt[2]

[1] Department of Mathematics, University of Britsh Columbia,
Vancouver, BC V6T 1Z2, Canada
jf@math.ubc.ca, www.math.ubc.ca/~jf
[2] Fakultät für Informatik, TU Chemnitz, 09107 Chemnitz, Germany
goerdt@informatik.tu-chemnitz.de,
www.tu-chemnitz.de/informatik/HomePages/TI

Abstract. It is known that random k-SAT instances with at least dn clauses where $d = d_k$ is a suitable constant are unsatisfiable (with high probability). This paper deals with the question to certify the unsatisfiability of a random 3-SAT instance in polynomial time. A backtracking based algorithm of Beame et al. works for random 3-SAT instances with at least $n^2/\log n$ clauses. This is the best result known by now.
We improve the $n^2/\log n$ bound attained by Beame et al. to $n^{3/2+\varepsilon}$ for any $\varepsilon > 0$. Our approach extends the spectral approach introduced to the study of random k-SAT instances for $k \geq 4$ in previous work of the second author.

Introduction

We study the complexity of certifying unsatisfiability of random 3-SAT instances (or 3-CNF formulas) over n propositional variables. The probability space of random 3-SAT instances has been widely studied in recent years for several good reasons. The most recent literature is [Ac2000], [Fr99], [Be et al97].

One of the reasons for studying random 3-SAT instances is that they have the following sharp threshold behaviour [Fr99]: There exist values $c = c(n)$ such that for any $\varepsilon > 0$ formulas with at most $(1-\varepsilon) \cdot c \cdot n$ clauses are satisfiable whereas formulas with at least $(1+\varepsilon) \cdot c \cdot n$ are unsatisfiable with high probability (that means with probability tending to 1 when n goes to infinity). Note, that the aforementioned result does not say that $c = c(n)$ is a constant, however the general conjecture is that $c(n)$ converges to a constant. Much recent work tries to approximate $c(n)$ and the currently best results are that $c(n)$ is at least 3.125 [Ac2000] and at most 4.601 [KiKrKr98]. Inaccessible to the authors at the time of writing is a FoCS 2000 paper making further progress on the lower bound for $c(n)$. (For random 2-SAT instances the analogous threshold is at $c = 2$ [ChRe92], [Go96].)

The algorithmic interest in this threshold is due to the empirical obeservation that random 3-SAT instances at the threshold, i.e. with around $c \cdot n$ random

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 310–321, 2001.

clauses are hard instances. The following behaviour has been reported consistently in experimental studies with backtracking algorithms for satisfiability, see for example [SeMiLe96] [CrAu96]: The average running time is quite low for instances below the threshold. For instances with at most $4n$ clauses the formulas are satisfiable and it is quite easy to find a satisfying assignment. A precipitous increase in the average complexity is observed at the threshold. For $4.2n$ clauses about half of the generated formulas are satisfiable and it is difficult to decide if a formula is satisfiable or not. Finally a speedy decline to lower complexity is observed beyond the threshold. All instances with $4.5n$ clauses are unsatisfiable and the running time decreases again (in spite of the fact that now the whole backtracking tree must be searched.) Note however that the decline in running time cannot yield polynomial average time. This follows from the paper [ChSz88] on which we comment further below.

Except of trivial observations there are no general complexity theoretical results relating the threshold to hardness. The relationship of hardness and thresholds has also been observed for k-colourability of random graphs with a linear number of edges [PeWe89], [AcFr99] and for the subset sum problem, see [ImNa96] for a theoretical discussion.

Abandoning the general point of view and looking at concrete algorithms the following results are known for random 3-SAT instances: All progress approximating the threshold from below is based on the analysis of rather simple polynomial time heuristics. In fact the most advanced heuristic being analyzed [Ac2000] only finds a satisfying assignment with probability of at least ε where $\varepsilon > 0$ is a small constant for 3-SAT formulas with at most $3.145n$ clauses. The heuristic in [FrSu96] finds a satisfying assignment for 3-SAT almost always for 3-SAT instances with at most $3.003n$ clauses. On the other hand the progress made in approximating the threshold from above does not provide us at all with efficient algorithms. Here only the expectation of the number of satisfying assignments is calculated and is shown to tend to 0.

In fact, beyond the threshold we have negative results: For arbitrary but fixed d beyond the threshold random 3-SAT instances with dn clauses (are unsatisfiable and) have only resolution proofs with an exponential number, that is with at least $2^{\Omega(n)}$ clauses [ChSz88]. This has been improved upon by [Fu98], [BePi96], and [Be et al97] all proving (exponential) lower bounds for larger clause/variable ratios. Note that the size of resolution proofs is a lower bound on the number of nodes in any classical backtracking tree as generated by any variant of the well known Davis-Putnam procedure.

Next we come to the historical development of polynomial time results beyond the threshold. In [Fu98] it is shown that 3-SAT formulas with at least n^2 clauses allow for polynomial size resolution proofs. This is strengthened in [Be et al97] to the best result known by now: For random 3-SAT instances with at least $n^2/\log n$ clauses a backtracking based algorithm proves unsatisfiability in polynomial time with high probability. (The result of Beame et al. is slightly stronger as is applies to formulas with $\Omega(n^2/\log n)$ clauses.) For general k-SAT the algorithm of Beame et al. works for formulas with at least $n^{k-1}/(\log n)^{k-2}$

clauses. This is improved in [GoKr2000] where a spectral approach is shown to work for at least $\mathrm{poly}(\log n) \cdot n^{k/2}$ random clauses for even k. For odd k we get the exponent $(k+1)/2$ instead of $k/2$. This implies that for $k = 3$ the result of Beame et al. is still the best known.

We extend the approach of our previous paper to show that for any $\varepsilon > 0$ random 3-SAT instances with at least $n^{3/2+\varepsilon}$ clauses can be efficiently certified as unsatisfiable thus improving the previous best bound of Beame et al. As in [GoKr2000] we associate a graph with a given formula. Then we show how to certify unsatisfiability of the formula with the help of the eigenvalue spectrum of the adjacency matrix of this graph. Note that the eigenvalue spectrum can be approximated with sufficient precision in polynomial time by standard linear algebra methods.

One technical contribution of the present paper when compared to [GoKr2000] is a lemma bounding the size of the largest independent set of vertices of a graph with the help of the eigenvalue gap of the adjacency matrix of this graph. We speculate that this may be of independent interest. In [GoKr2000] we use a matrix derived from the adjacency matrix instead of the adjacency matrix itself to bound the size of the largest independent set. In [Ch97] bounds on the size of the largest independent set are given in terms of the spectral gap of the Laplacian matrix of the graph. These results cannot be directly applied to the present situation because in our case it is not clear how to estimate the eigenvalues of the Laplacian matrix, instead of those of the adjacency matrix.

Note that eigenvalues can be used to help to find a solution to a random instance of an NP-complete problem [AlKa94] or to prove the absence of a solution as in our case, for example.

1 From Random 3-SAT Instances to Random Graphs

We consider a family probability spaces of random 3-SAT instances, $\mathrm{Form}_{n,p} = \mathrm{Form}_{n,p,3}$, which is defined as follows: The set of 3-clauses is the set of all 3-tuples $l_1 \vee l_2 \vee l_3$ where l_i is a literal over a standard set of n propositional variables. A literal either is a propositional variable x or its negation $\neg x$. As double occurrences of literals inside of clauses are allowed we have altogether $(2n)^3 = 8n^3$ 3-clauses. A random instance F from $\mathrm{Form}_{n,p}$ is obtained by adding each single clause independently with probability p to F. We think of the clauses of F as being joined conjunctively and write $F = \{C_1, \ldots, C_m\} = C_1 \wedge \ldots \wedge C_m$. In the sequel we assume that $p = p(n) = 1/n^{1+\gamma}$ where $1/2 > \gamma > 0$ is a constant. Note that our space of formulas is analogous to the space of random graphs $G_{n,p}$. The number of clauses in a random instance from $\mathrm{Form}_{n,p}$ follows the binomial distribution with parameters $8n^3$ and p, $\mathrm{Bin}(8n^3, p)$ and the expected number of clauses is $8n^3 \cdot p = 8n^{2-\gamma} > n^{3/2}$.

Another popular family of probability spaces of random 3-SAT instances is the space $\mathrm{Form}_{n,m}$. Here each formula is a set of exactly m clauses and each formula has probability $1/\binom{8n^3}{m}$. $\mathrm{Form}_{n,m}$ is analogous to the space of random

graphs $G_{n,m}$. We feel confident and in line with common usage that our results also apply to $\text{Form}_{n,m}$ if $m = 8n^3 \cdot p$ is the expected number of clauses in our $\text{Form}_{n,p}$ model. See [Bo85] for a general theorem relating analogous random graph models. Clauses might be defined in a slightly different way, too: They might be sets of literals, they might be tuples without repetitions, they might be non-tautological, that is not containing x and $\neg x$ together. Again we assume without actually checking that our results apply to the other probability spaces, too.

We state a graph theoretical condition which implies the unsatisfiability of a 3-SAT instance F over n propositional variables. To this end we define the graphs G_F and H_F. The graph $G_F = (V_F, E_F)$ is defined as follows:

- V_F is the set of ordered pairs over the n propositional variables. We have $|V_F| = n^2$.
- The edge (a_1, b_1)——(a_2, b_2) (where in order to avoid loops $(a_1, b_1) \neq (a_2, b_2)$ that is $a_1 \neq a_2$ or $b_1 \neq b_2$) is in E_F if there exists a variable z such that F contains both clauses $a_1 \vee a_2 \vee z$ and $b_1 \vee b_2 \vee \neg z$ (or both clauses $a_2 \vee a_1 \vee z$ and $b_2 \vee b_1 \vee \neg z$, note however that our graph is undirected and it is not strictly necessary to mention explicitly this possibiliy.) Note that the a_i and b_i are variables, that is positive literals.

The graph H_F is defined totally analogously but with different clauses: Its vertices are as before ordered pairs of variables and (a_1, b_1)——(a_2, b_2) is an edge iff F has the clauses $\neg a_1 \vee \neg a_2 \vee z$ and $\neg b_1 \vee \neg b_2 \vee \neg z$ for a variable z.

Some comments concerning the intuition behing this definition: In [GoKr2000] we give an efficient algorithm which demonstrates the unsatisfiability of random 4-SAT instances with at least $\text{poly}(\log n) \cdot n^2$ clauses. Here we build on the techniques introduced in this paper. The clause $a_1 \vee a_2 \vee b_1 \vee b_2$ is obtained by resolution [Sch] with z from the two clauses $a_1 \vee a_2 \vee z$ and $b_1 \vee b_2 \vee \neg z$ which define an edge of G_F. Similarly we have that $\neg a_1 \vee \neg a_2 \vee \neg b_1 \vee \neg b_2$ is obtained from $\neg a_1 \vee \neg a_2 \vee z$ and $\neg b_1 \vee \neg b_2 \vee \neg z$. The correctness of resolution states that F is unsatisfiable if a set of resolvents of F is unsatisfiable.

For any given z the number of clauses like $a_1 \vee a_2 \vee z$ and $b_1 \vee b_2 \vee \neg z$ is concentrated at the expectation $\approx n^2 \cdot p = n^{1-\gamma} > n^{1/2}$ Applying resolution with z to all these clauses gives $\approx n^{(1-\gamma)2} > n$ clauses $a_1 \vee a_2 \vee b_1 \vee b_2$. Doing this for all n variables z gives $> n^2$ all-positive clauses of size 4. In the same way we get $> n^2$ all-negative 4-clauses. With the help of the technique introduced in [GoKr2000] we get an efficient algorithm which demonstrates unsatisfiability of these newly obtained 4-clauses and the correctness of resolution implies that F itself is unsatisfiable. Note that our graphs G_F and H_F reflect the sets of clauses obtained by resolution as above in that each clause induces an edge in one of these graphs.

Some detailed remarks concerning G_F: Only for technical reasons the variable z which is resolved upon is the *last* variable in our clauses. (Recall we consider clauses as ordered triples.) More important is the fact that the edge reflects the resolvent $a_1 \vee a_2 \vee b_1 \vee b_2$ *not* in the most natural way by the edge

(a_1, a_2)——(b_1, b_2) but by (a_1, b_1)——(a_2, b_2). The variables of the vertices connected by the edge come from the *different* clauses taking part in the resolution. The reason why this is important is to increase the independence of the edges of G_F when F is a random formula. Again more of a technical nature is the convention that the variables in the first position of each vertex come from the clause which contains the positive literal z, whereas the second variables b_1, b_2 come from the clause with $\neg z$.

Recall that a set S of vertices of a graph G is an *independent set* iff there is no edge inside of S, and $\alpha(G)$ is the *independence number* of G that is the maximum number of vertices of an independent set of G. The independence number is $\mathcal{NP}$-hard to determine. Therefore we cannot use the following theorem directly to get an efficient algorithm certifying unsatisfiability by simply computing the independence number of G_F and H_F. The proof of the next theorem relies on the correctness proof of resolution [Sch] and is not difficult.

Theorem 1. *If F is a 3-SAT instance over n variables which is satisfiable then we have:*

$$\alpha(G_F) \geq n^2/4 \quad or \quad \alpha(H_F) \geq n^2/4.$$

Within G_F (and H_F) the presence or absence of an edge is not independent of that of another edge, and so techniques from the area of standard random graphs cannot be applied without further consideration. From now on we restrict attention to G_F, of course everything applies also to H_F. We collect some basics about G_F.

An edge (a_1, b_1)——(a_2, b_2) in G_F is only possible if $a_1 \neq a_2$ or $b_1 \neq b_2$. We take a look at the structure of the clause sets which induce the fixed edge (a_1, b_1)——(a_2, b_2). The edge (a_1, b_1)——(a_2, b_2) is in G_F iff F contains for a variable z at least one of the pairs of clauses $a_1 \vee a_2 \vee z$ and $b_1 \vee b_2 \vee \neg z$ (or one of the pairs $a_2 \vee a_1 \vee z$ and $b_2 \vee b_1 \vee \neg z$).

Case 1: $a_1 \neq a_2$ and $b_1 \neq b_2$. In this case all the "z-clauses" necessary to induce the edge are distinct and all $\neg z$-clauses, too. As the z and $\neg z$ clauses are all distinct from each other, too, we have $2n$ disjoint pairs of clauses which induce the edge (a_1, b_1)——(a_2, b_2).

Case 2: $a_1 = a_2$ and $b_1 \neq b_2$. In this case the clauses $a_1 \vee a_2 \vee z$ necessary for the edge are all distinct. However $a_1 \vee a_2 \vee z = a_2 \vee a_1 \vee z$. The $\neg z$-clauses are all distinct and also the z- and $\neg z$-clauses. We have altogether $2n$ pairs of clauses where always two pairs have the common clause $a_1 \vee a_2 \vee z$. The last case $a_1 \neq a_2$ and $b_1 = b_2$ is analogous to the second case.

With these observations we can get a first impression of the probability of a fixed edge in G_F: If $a_1 \neq a_2$ and $b_1 \neq b_2$ the number of pairs of clauses which induce the edge (a_1, b_1)——(a_2, b_2) is distributed as $\mathrm{Bin}(2n, p^2)$. The probability that the edge is induced by two pairs of clauses is at most $\binom{2n}{2} \cdot p^4 = o(2np^2)$. This makes it intuitively clear that the probability of (a_1, b_1)——(a_2, b_2) being in G_F is about $2n \cdot p^2$.

If $a_1 = a_2$ and $b_1 \neq b_2$ we observe that the number of clauses like $b_1 \vee b_2 \vee \neg z$ or $b_2 \vee b_1 \vee \neg z$ is distributed as $\mathrm{Bin}(2n, p)$. The probability to have at

least two of these clauses is $o(2np)$. Conditioning on the occurrence of at least one of these clauses it becomes inuitively clear that the probability of the edge (a_1, b_1)——(a_2, b_2) should also be about $2n \cdot p^2$. The following two results make this precise.

Lemma 1. *We fix the edge* $e = (a_1, b_1)$——(a_2, b_2) *and recall* $p = 1/n^{1+\gamma}$.
(a) For $a_1 \neq a_2$ *and* $b_1 \neq b_2$ *we have that*

$$Pr[G_F;\ e \ \textit{is an edge of}\ G_F] \ = \ 2n \cdot p^2 \cdot (1 \ + \ O(\frac{1}{n^{1+2\gamma}})).$$

(b) For $a_1 = a_2$ *and* $b_1 \neq b_2$ *this probability is*

$$2n \cdot p^2 \cdot (1 \ + \ O(\frac{1}{n^{1+\gamma}})).$$

The same applies of course to $a_1 \neq a_2$ *and* $b_1 = b_2$.

The preceding lemma implies the following expectations:

Corollary 1. *(a)* $E[\textit{Number of edges of}\ G_F] \ = \ n^{3-2\gamma} \cdot (1 \ + \ O(\frac{1}{n}))$.
(b) $E[\textit{Degree of the vertex}\ \ (a_1, b_1)] \ = \ 2n^{1-2\gamma} \cdot (1 + O(\frac{1}{n}))$.

Observe that $n^2 \cdot 2n^{1-2\gamma} \ = \ 2 \cdot n^{3-2\gamma}$ reflecting the fact that the sum of the degrees of all vertices is two times the number of edges. The number of vertices altogether is equal to n^2 and the probability of a given edge is $\approx 2/n^{1+2\gamma}$. Disregarding edge dependencies G_F is a random graph $G_{n^2,p'}$ where $p' = 2/n^{1+2\gamma}$. As $\gamma < 1/2$ this situation is equivalent to that of a random graph over n vertices with edge probability n^δ/n with $\delta > 0$.

2 Concentration of the Degree

The degree of a given vertex of a random graph with n vertices and edge probability n^δ/n follows the binomial distribution $\mathrm{Bin}(n-1, n^\delta/n)$. Exponential tail bounds for the binomial distribution imply that each vertex has its degree sharply concentrated at its expectation $\approx n^\delta$. (Note that $\exp(-n^\delta) \ = \ o(1/n)$ and we can proceed from a fixed vertex to *all* vertices.) To show an analogous result for G_F we consider a fixed vertex (a_1, b_1) and determine the number of edges like (a_1, b_1)——(a_2, b_2). Before looking at the degree of (a_1, b_1) itself we look at the number of unordered pairs of clauses

$$a_1 \vee a_2 \vee z \text{ and } b_1 \vee b_2 \vee \neg z \tag{1}$$

in a random F where a_2, z, b_2 are arbitrary. To show concentration of the number of pairs of clauses as in (1) we follow [AlSp92, Chapter 8, Section 4. The technical problem to deal with is that distinct pairs are not always disjoint and thus are not independent. Nevertheless the intuition is that they are nearly independent and we have:

Theorem 2. *Let $\varepsilon > 0$ be fixed, let X be the random variable giving number of pairs of clauses as in (1) and let μ be the expectation of X. Then $Pr[|X - \mu| > \varepsilon\mu] = o(1/n^2)$.*

Some pairs of clauses as in (1) might induce the same edge. But that does not destroy concentration:

Corollary 2. *Let a_1 and b_1 be two variables and $\varepsilon > 0$. The degree of the vertex (a_1, b_1) in G_F is with probability $1 - o(1/n^2)$ between $2n^{1-2\gamma}(1-\varepsilon)$ and $2n^{1-2\gamma}(1+\varepsilon)$. Moreover, with high probability the degree of* each *vertex is within the specified interval.*

3 Spectral Considerations

In this section we prove a general relationship between the size of an independent set in a graph and the eigenvalues of its adjacency matrix. Then we prove that the random graphs G_F and H_F satisfy certain eigenvalue bounds with high probability. These eigenvalue bounds certify that the graphs G_F and H_F do not have independent sets as required by Theorem 1 in order for F to be satisfiable. Background from spectral graph theory can be found for regular graphs in [AlSp92] and for the general case in [Ch97]. The linear algebra required is well presented in [St88].

Let $G = (V, E)$ be a standard undirected graph and A_G the adjacency matrix of G. Let A_G's eigenvalues be ordered $\lambda_1 \geq \cdots \geq \lambda_n$, with $n = |V|$. We say that G is *ν-separated* if $|\lambda_i| \leq \nu\lambda_1$ for $i > 1$. With $\lambda = \max_{i>1} |\lambda_i|$ this reads $\lambda \leq \nu\lambda_1$. We say that G is *ϵ-balanced* for some $\epsilon > 0$ if there is a real d such that the degree of each vertex is between $d(1-\epsilon)$ and $d(1+\epsilon)$.

Theorem 3. *If G is ν-separated and ϵ-balanced, then G contains no independent set of size $> (n/5) + n \cdot f(\nu, \epsilon)$ where $f(\nu, \epsilon)$ tends to 0 as ν, ϵ tend to 0.*

We remark that this theorem can probably be greatly improved upon. But this weak theorem does preclude independent sets of size $n/4$ for small ν, ϵ, and that is all we need here.

Proof. Let S be an independent subset of vertices of G. We will bound $|S|$. Let $T = V \setminus S$. Let χ_S, χ_T be the characteristic functions (represented as column vectors) of S, T respectively (i.e. taking the value 1 inside the set and 0 outside the set). As S is an independent set and G is ε-balanced, we have

$$d(1-\epsilon)|S| \leq \Big| \text{ edges leaving } S \Big| = < A_G\chi_S, \chi_T > . \tag{2}$$

Note that $A_G\chi_S$ is the column vector whose i'th entry is the number of edges going from vertex i into the set S. Recall that $T = V \setminus S$ and $< \cdots, \cdots >$ is the standard inner product of two vectors. We show further below that

$$< A_G\chi_S, \chi_T > \leq \; d(1+\varepsilon) \cdot (1/2 \; + \; \nu) \cdot \sqrt{|S||T|}. \tag{3}$$

Abbreviating $\theta = |S|/n$ we get from (2) and (3) that
$\theta/(1-\theta) \leq (1/2 + g(\nu,\varepsilon))^2 = 1/4 + g(\nu,\varepsilon) + g(\nu,\varepsilon)^2$ where
$g(\nu,\varepsilon) = (\varepsilon+(1+\varepsilon)\nu)/(1-\varepsilon)$ as can be seen by elementary algebra. Note that $g(\nu,\varepsilon)$ goes to 0 when ν and ε approach 0. We set $f(\nu,\varepsilon) = (4/5)(g+g^2)$ and get: $\theta \leq (1/4)(4/5) + (4/5)(g+g^2) = 1/5 + f$, which is the theorem.

We need to show inequality (3). Let $u_1,\ldots,u_n$ be an orthonormal basis of the n-dimensional vectorspace over the reals where u_i is an eigenvector of A_G with eigenvalue λ_i. We can decompose the adjacency matrix as
$A_G = \lambda_1 \cdot u_1 \cdot u_1^T + \lambda_2 \cdot u_2 \cdot u_2^T + \cdots + \lambda_n \cdot u_n \cdot u_n^T$, where $u_i^T = (u_{i,1},\ldots,u_{i,n})$ is the transpose of the column vector u_i. Note that $\lambda_i \cdot (u_i \cdot u_i^T) \cdot v = \lambda_i \cdot v$ if $v = \alpha \cdot u_i$ and $\lambda_i \cdot (u_i \cdot u_i^T) \cdot v = 0$ for v orthogonal to u_i. Let $\mathcal{E} = A_G - \lambda_1 \cdot u_1 \cdot u_1^T = \sum_{i\geq 2} \lambda_i \cdot u_i \cdot u_i^T$. and represent χ_S, χ_T over the basis of the u_i: $\chi_S = \sum_{i=1}^n \alpha_i \cdot u_i$ and $\chi_T = \sum_{i=1}^n \beta_i \cdot u_i$. Recall here the fact known as Parseval's equation: $|S| = ||\chi_S||^2 = \sum \alpha_i^2$ and $|T| = ||\chi_T||^2 = \sum \alpha_i^2$. We get easily
$< A_G\chi_S, \chi_T > = (\lambda_1(u_1^T \cdot \chi_S)) \cdot (u_1^T \cdot \chi_T) + < \mathcal{E} \cdot \chi_S\,,\, \chi_T >$ and proceed to bound both summands separately.

Because of the orthornormality of the u_i we get:

$$< \mathcal{E}\chi_S\,,\, \chi_T > \leq \lambda \cdot \sqrt{\sum_{i\geq 1} \alpha_i^2} \cdot \sqrt{\sum_{i\geq 1} \beta_i^2} = \lambda \cdot \sqrt{|S|} \cdot \sqrt{|T|} \leq \nu \cdot d(1+\varepsilon)\sqrt{|S||T|}$$

where the last step holds because λ_1 is bounded above by the maximum degree of the vertices, the last but first step uses Parseval's equation and the last but second Cauchy-Schwarz inequality, $\sum |\alpha_i\beta_i| \leq \sqrt{\sum \alpha_i^2} \cdot \sqrt{\sum \beta_i^2}$.

Now we come to the other summand, $(\lambda_1(u_1^T \cdot \chi_S)) \cdot (u_1^T \cdot \chi_T)$. Let α, β be the average values of u_1 on S, T respectively, that is $\alpha = (\sum u_{1,j})/|S|$ where the sum goes over $j \in S$. With the inequality of Cauchy-Schwarz we get:

$$\alpha^2 = \frac{(\sum (u_{1,j} \cdot 1))^2}{|S|^2} \leq \frac{(\sum u_{1,j}^2) \cdot (\sum 1)}{|S|^2} = \frac{\sum u_{1,j}^2}{|S|}$$

which implies $\alpha^2|S| \leq \sum u_{1,j}^2$. As $T = V \setminus S$ we get

$$\alpha^2|S| + \beta^2|T| \leq \sum_{j=1}^{n} u_{1,j}^2 = ||u_1||^2 = 1.$$

Using the fact that the geometric mean is bounded by the arithmetic mean this implies $\alpha\sqrt{|S|} \cdot \beta\sqrt{|T|} \leq 1/2$. (The weakness of this theorem undoubtedly comes from the pessimistic estimate used $\alpha\sqrt{|S|} \cdot \beta\sqrt{|T|} \leq (\alpha^2|S| + \beta^2|T|)/2$ which is only close to the truth when $\alpha^2|S|$ is close to $\beta^2|T|$). This implies as λ_1 is bounded above by the maximum degree that

$$\lambda_1 \cdot (u_1^T\chi_S) \cdot (u_1^T\chi_T) = d(1+\varepsilon)\alpha|S| \cdot \beta|T| \leq (1/2) \cdot d(1+\varepsilon)\sqrt{|S||T|}$$

and we get (3) finishing the proof. □

We next show that the graphs G_F, and H_F are ν-separated for a small ν. We do this by applying the trace method, see for example [Fr91], in an elementary form. We first give a general outline of this method. For $A = A_G$ an adjacency

matrix we have from linear algebra that for each $k \geq 0$ $\mathrm{Trace}(A^k) = \sum_{i=1}^{n} \lambda_i^k$. (The trace of a matrix is the sum of the elements on the diagonal.) TraceA^k can be calculated from the underlying graph as: $\mathrm{Trace}(A^k) = \left|\right.$ closed walks of length k in the underlying graph $\left.\right|$. A closed walk of length k in G is a walk like $a_0 \overset{e_1}{\text{———}} a_1 \overset{e_2}{\text{———}} a_2 \overset{e_3}{\text{———}} \cdots \overset{e_{k-1}}{\text{———}} a_{k-1} \overset{e_k}{\text{———}} a_k = a_0$. Note that the e_i and a_i need by no means be distinct. As we assume the graph loopless we can only conclude that $a_{i-1} \neq a_i$. For all even k we have that all $\lambda_i^k \geq 0$ and we get $\mathrm{Trace}(A^k) = \sum_{i=1}^{n} \lambda_i^k \geq \lambda_1^k + \max_{i>1} \lambda_i^k$. Abbreviating $\lambda = \max_{i>1} \lambda_i$ we get further $\lambda^k \leq \sum_{i=2}^{n} \lambda_i^k$. If the underlying matrix A is the adjacency matrix of a random graph this applies in particular to the expected values:

$$E[\lambda^k] \leq E[\mathrm{Trace}(A^k)] - E[\lambda_1^k] = E[\sum_{i=2}^{n} \lambda_i^k]. \tag{4}$$

Now assume that we have an underlying variable n as in the $G_{n,p}$ model of random graphs and that $E[\lambda^k] = o(\lambda_1^k)$ holds with high probability. Then for each constant $\nu > 0$ we get $Pr[\lambda > \nu\lambda_1] = Pr[\lambda^k > (\nu\lambda_1)^k] =$
$= Pr\left[\lambda^k > ((\nu\lambda_1)^k / E[\lambda^k]) \cdot E[\lambda^k]\right] \leq Pr\left[\lambda^k > 1/o(1) \cdot E[\lambda^k]\right] + o(1) \leq o(1)$
where we apply Markov's inequality and use the fact that ν and k are constant. This says that the graphs considered are ν-separated with high probability. (The idea considering the k-th power of the eigenvalues seems to be to increase the gap between the largest eigenvalue and the remaining eigenvalues.) The proof of the following lemma prepares for the more complex situation with the graphs G_F and H_F.

Lemma 2. *Let $\nu > 0$ and $\delta > 0$ be constants. With high probability a random graph from $G_{n,p}$ with $p = n^\delta/n$ is ν-separated.*

Proof. Let A be the adjacency matrix of a random graph from $G_{n,p}$. Let k be an even constant to be specified further below. We bound $E[\lambda^k]$ by bounding $E[\mathrm{Trace}(A^k)] - E[\lambda_1^k]$, see (4). We calculate both expectations separately. From concentration of the degree of each vertex we get that with high probability λ_1 is between $(1-\varepsilon)n^\delta$ and $(1+\varepsilon)n^\delta$. As the probability of failure is exponentially low in n and k is constant we get $E[\lambda_1^k] \geq (1-\varepsilon)^k n^{\delta k} - o(1)$. Next we come to the expectation of the trace. For $\boldsymbol{a} = (a_0, \ldots, a_{k-1}, a_k = a_0)$ let walk($\boldsymbol{a}$) be the indicator random variable of the event that the walk given by $\boldsymbol{a}$ is possible in a random graph, that is all edges $e_i = (a_{i-1}, a_i) = \{a_{i-1}, a_i\}$ for $1 \leq i \leq k$ occur. Then $E[\mathrm{Trace}(A^k)] = \sum_{\boldsymbol{a}} P[\mathrm{walk}(\boldsymbol{a}) = 1]$. To calculate the preceding sum we distinguish three types of possible walks $\boldsymbol{a}$. A walk is *distinct* iff all edges e_i are distinct. A walk is *duplicated* iff each edge among the e_i occurs at least twice. A walk is *quasi-distinct* iff some edges among the e_i occur at least twice and

some only once. For $\boldsymbol{a}$ distinct we have that the expected number of such walks is bounded above by $n^{\delta k}$. (Compare to our estimate for $E[\lambda_1]$ above.)

For $\boldsymbol{a}$ duplicated we parameterize further with respect to the number j with $1 \leq j \leq k/2$, of distinct edges among the e_i. the number of possibilities here is at most $n^{j+1}k^{2k}$ and for the expected number of duplicated walks we get the upper bound $\sum_{j=1}^{k/2} k^{2k} \cdot n^{j+1} \cdot (n^{\delta}/n)^j \leq \sum_{j=1}^{k/2} k^{2k} \cdot n \cdot n^{\delta j} \leq (k/2) \cdot k^{2k} \cdot n \cdot n^{\delta k/2}$. Note that picking $k > 2/\delta$ implies that $1 + \delta k/2 < \delta k$ which in turn implies that the bound is $o(n^{\delta k})$. Note that we must pick k larger when δ gets smaller in order that the last statement holds. (This is reassuring.)

For the number of quasi-distinct walks we first assume that the last edge, e_k, is a unique edge of the walk. We get similarly to the preceding bound a bound of $k \cdot k^{2k} \cdot n^{\delta(k-1)}$. As the last edge need not always be unique we get an additional factor of k. The estimate is always $o(n^{\delta k})$ as $\delta(k-1) < \delta k$. Summing all preceding estimates we get $E[\mathrm{Trace}(A^k)] \leq n^{\delta k} + o(n^{\delta k})$ and $E[\lambda^k] \leq$ $\leq (1-(1-\varepsilon)^k) \cdot n^{\delta k} + o(n^{\delta k})$. As $\varepsilon > 0$ can be chosen arbitrarily small, k is constant and the preceding estimate holds whenever n is sufficiently large this means that $E[\lambda^k] = o(n^{\delta k}) = o(\lambda_1^k)$ with high probability. By the general principle above the graphs from $G_{n,p}$ are ν-separated with high probability. □

As graphs from $G_{n,p}$ are ε-balanced for any $\varepsilon > n$ Theorem 3 implies that we can efficiently certify that a random graph from $G_{n,p}$ has no independent set with much more than $n/5$ vertices with high probability. The treatment of our graphs G_F, H_F based on the method above is more technical but otherwise the same.

Theorem 4. *For $F \in Form_{n,p,3}$ let A_F be the adjacency matrix of G_F and let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{n^2}$ be the eigenvalues of A_F. Then $E\left[\sum_{i=1}^{n^2} \lambda_i^k\right]$ is equal to*

$$E[\mathit{Trace}(A_F^k)] \leq \left(2n^{1-2\gamma}\right)^k + c \cdot k^4 \cdot k^{4k} \cdot 2^k \cdot (n^{(1-2\gamma)(k-1)} + n^2 \cdot n^{(1-2\gamma)k/2}),$$

where c is a constant ($c = 100$ should be enough). If $k > 4/(1-2\gamma)$ the preceding estimate is $\left(2n^{1-2\gamma}\right)^k + o\left(\left(2n^{1-2\gamma}\right)^k\right)$ (compare Corollary 2). The same applies to H_F.

Proof. For any F we have that $\mathrm{Trace}(A_F) = \left|\text{closed walks of length } k \text{ in } G_F\right|$. A typical closed walk of length k is (a_0, b_0)——$(a_1, b_1) \cdots$ ——(a_{k-1}, b_{k-1}) ——$(a_k, b_k) = (a_0, b_0)$. Now consider a step (a_{i-1}, b_{i-1})——(a_i, b_i) of this walk. For this step to be possible in G_F the formula F must have one of the following $2n$ pairs of clauses: $a_{i-1} \vee a_i \vee z$, $b_{i-1} \vee b_i \vee \neg z$ for a propositional variable z or the other way round, that is a_i, b_i first. We say that pairs of the first type induce the step (a_{i-1}, b_{i-1})——(a_i, b_i) with sign $+1$ whereas the second type induces this step with sign -1. For two sequences of clauses $\boldsymbol{C} = (C_1, C_2, \ldots, C_k)$ where the

last literal of each C_i is a positive literal and $\boldsymbol{D} = (D_1, D_2, \ldots, D_k)$, where the last literal of each D_i is negative, and a a sequence of signs $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_k)$ we say that $\boldsymbol{C}, \boldsymbol{D}, \boldsymbol{\varepsilon}$ induce the walk above iff for each i the pair of clauses C_i, D_i induces the i'th step of the walk with sign given by ε_i. Note that the occurrences of the clauses D_i and C_j in a random F are independent as these clauses are always distinct. We say that F induces the walk above iff we can find sequences of clauses $\boldsymbol{C}, \boldsymbol{D} \subseteq F$ (the C_i, D_i need not necessarily be all distinct) and a sequence of signs $\boldsymbol{\varepsilon}$ such that $\boldsymbol{C}, \boldsymbol{D}, \boldsymbol{\varepsilon}$ induce the given walk. We observe: First, G_F allows for a given walk iff F induces this walk as defined above. Second, three sequences $\boldsymbol{C}, \boldsymbol{D}, \boldsymbol{\varepsilon}$ induce at most one walk, but one walk can be induced by many $\boldsymbol{C}, \boldsymbol{D}, \boldsymbol{\varepsilon}$'s. (Without the $\boldsymbol{\varepsilon}$ it is possible that $\boldsymbol{C}, \boldsymbol{D}$ induce several walks.) Thus we get that $\mathrm{Trace}(A_F^k)$ can be bounded above by $\left|\boldsymbol{C}, \boldsymbol{D}, \boldsymbol{\varepsilon}' s, \text{ with } \boldsymbol{C}, \boldsymbol{D} \subseteq F \text{ inducing a closed walk of length } k\right|$ and this transfers to the expectations over random formulas F. The notions distinct, quasi-distinct, and duplicated transfer naturally from graphs to $\boldsymbol{D}$, $\boldsymbol{C}$. We decompose the expected number of $\boldsymbol{C}, \boldsymbol{D}, \boldsymbol{\varepsilon}$'s which induce a closed walk of length k according to all combinations of $\boldsymbol{C}, \boldsymbol{D}$ being distinct, quasi-distinct or duplicated. The reader with some experience can now easily fill in the remaining technical detail alogn the lines of the proof of Lemma 2. □

Now our algorithm is obvious: We pick ε, ν sufficiently small such that the $f(\nu, \varepsilon)$ from Theorem 3 is $< 1/20$ (because $1/5 + 1/20 = 1/4$). Given $F \in \mathrm{Form}_{n,p}$ where $p = 1/n^{1+\gamma}$ we construct G_F. Corollary 2 and Theorem 4 imply that G_F is ε-balanced and ν-separated with high probability. We efficiently check if maximum degree/minimum degree $\leq (1+\varepsilon)/(1-\varepsilon)$. This holds with high probability, in case it does not the algorithm fails. Now we determine λ_1 and λ with sufficient precision. We have that $\lambda \leq \nu\lambda_1$ with high probability. If the last estimate does not hold, we fail. By Theorem 3 the algorithm now has certified that G_F has no independent set of size $\geq n^2/4$. We do the same for H_F. With high probability we succeed and by Theorem 1 F is certified unsatisfiable.

Our algorithm works with high probability with respect to the binomial space $\mathrm{From}_{n,p}$ where p is such that the *expected number* of clauses is the announced $n^{3/2+\varepsilon}$. In case we want to show that it works for the space $\mathrm{Form}_{n,m}$ with $m = n^{3/2+\varepsilon}$ additional consideration is necessary: We would have to show that the algorithm fails in $\mathrm{Form}_{n,p}$ only with probability of $o(1/\sqrt{n})$. This is sufficient because the Local Limit Theorem implies that the set of formulas in $\mathrm{Form}_{n,p}$ having exactly the expected number of clauses has probability bounded below by $\Omega(1/\sqrt{n})$. We leave the detailed asymptotics required (we guess that k must go slowly to infinity for this) to the full version.

Acknowledgement. Helpful and detailed remarks of a referee improve presentation.

References

[Ac2000] Dimitris Achlioptas. Setting 2 variables at a time yields a new lower bound for random 3-SAT. In Proceedings SToC 2000.

[AcFr99] Dimitris Achlioptas, Ehud Friedgut. A threshold for random k-colourability. Random Structures and Algorithms 1999.

[AlKa94] Noga Alon, Nabil Kahale. A spectral technique for colouring random 3-colourable graphs (preliminary version). In Proceedings SToC 1994. ACM. 346-355.

[AlSp92] Noga Alon, Joel H. Spencer. The probabilistic method. Wiley & Sons Inc. 1992.

[Be et al97] Paul Beame, Richard Karp, Toniann Pitassi, Michael Saks. On the complexity of unsatisfiability proofs for random k-CNF formulas. 1997.

[BePi96] Paul Beame, Toniann Pitassi. Simplified and improved resolution lower bounds. In Proceedings FoCS 1996. IEEE. 274-282.

[Bo85] Bela Bollobas. Random Graphs. Academic Press. 1985.

[Ch97] Fan R. K. Chung. Spectral Graph Theory. American Mathematical Society. 1997.

[ChRe92] Vasek Chvatal, Bruce Reed. Mick gets some (the odds are on his side). In Proceedings 33nd FoCS 1992. IEEE. 620-627.

[ChSz88] Vasek Chvatal, Endre Szemeredi. Many hard examples for resolution. Journal of the ACM 35(4), 1988, 759-768.

[CrAu96] J. M. Crawford, L. D. Auton. Experimental results on the crossover point in random 3SAT. Artificial Intelligence 81, 1996.

[Fr91] Joel Friedman. Combinatorica, 1991.

[Fr99] Ehud Friedgut. Necessary and sufficient conditions for sharp thresholds of graph properties and the k-SAT problem. Journal of the American Mathematical Society 12, 1999, 1017-1054.

[FrSu96] Alan M. Frieze, Stephen Suen. Analysis of two simple heuristics on a random instance of k-SAT. Journal of Algorithms 20(2), 1996, 312-355.

[Fu98] Xudong Fu. The complexity of the resolution proofs for the random set of clauses. Computational Complexity, 1998.

[Go96] Andreas Goerdt. A threshold for unsatisfiability. Journal of Computer and System Sciences 53, 1996, 469-486.

[GoKr2000] Andreas Goerdt, Michael Krivelevich. Efficient recognition of random unsatisfiable k-SAT instances by spectral methods. In Proceedings STACS 2001. LNCS.

[ImNa96] Russel Impagliazzo, Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. Journal of cryptology 9, 1996, 199-216.

[KiKrKr98] Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, Yiannis Stamatiou. Approximating the unsatisfiability threshold of random formulas. Random Structures and Algorithms 12(3), 1998, 253-269.

[PeWe89] A. D. Petford, Dominic Welsh. A randomised 3-colouring algorithm. Discrete Mathematics 74, 1989, 253-261.

[Sch] Uwe Schöning. Logic for Computer Science. Birkhäuser.

[SeMiLe96] Bart Selman, David G. Mitchell, Hector J. Levesque. Generating hard satisfiability problems. Artificial Intelligence 81(1-2), 1996, 17-29.

[St88] Gilbert Strang. Linear Algebra and its Applications. Harcourt Brace Jovanovich, Publishers, San Diego. 1988.

Weisfeiler-Lehman Refinement Requires at Least a Linear Number of Iterations

Martin Fürer*

Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802, USA
furer@cse.psu.edu,
http://www.cse.psu.edu/~furer

Abstract. Let $\mathcal{L}_{k,m}$ be the set of formulas of first order logic containing only variables from $\{x_1, x_2, \ldots, x_k\}$ and having quantifier depth at most m. Let $\mathcal{C}_{k,m}$ be the extension of $\mathcal{L}_{k,m}$ obtained by allowing counting quantifiers $\exists i x_j$, meaning that there are at least i distinct x_j's.
It is shown that for constants $h \geq 1$, there are pairs of graphs such that h-dimensional Weisfeiler-Lehman refinement (h-dim W-L) can distinguish the two graphs, but requires at least a linear number of iterations. Despite of this slow progress, $2h$-dim W-L only requires $O(\sqrt{n})$ iterations, and $3h-1$-dim W-L only requires $O(\log n)$ iterations. In terms of logic, this means that there is a $c > 0$ and a class of non-isomorphic pairs (G_n^h, H_n^h) of graphs with G_n^h and H_n^h having $O(n)$ vertices such that the same sentences of $\mathcal{L}_{h+1,cn}$ and $\mathcal{C}_{h+1,cn}$ hold ($h+1$ variables, depth cn), even though G_n^h and H_n^h can already be distinguished by a sentence of $\mathcal{L}_{k,m}$ and thus $\mathcal{C}_{k,m}$ for some $k > h$ and $m = O(\log n)$.

Keywords: Graph Isomorphism Testing, Weisfeiler-Lehman Refinement, Games, Descriptive Complexity

1 Introduction

A simple and important preprocessing procedure for the graph isomorphism problem is the k-dimensional Weisfeiler-Lehman refinement (k-dim W-L). The algorithm tries to color k-tuples of vertices with different colors, if they belong to different orbits of the automorphism group. This goal is not always achieved. If two k-tuples have the same color, it is still possible that no automorphism maps one to the other, but the algorithm has not discovered a significant difference between the two k-tuples. On the other hand, if two k-tuples have different colors, then they always belong to different orbits.

For $k = 1$, this is the straightforward vertex classification algorithm where vertices are initially colored by their degrees. During every later refinement step,

* Research supported in part by NSF Grant CCR-9700053

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 322–333, 2001.

each vertex is colored by the multi-set of the colors of its neighbors. The process stops, when no color class is split anymore.

The case $k = 2$ has also been well studied. It is edge coloring. The algorithm starts with three classes of pairs of vertices: pairs (u, v) with or without an edge, and pairs (u, u). During each refinement step, every directed edge (u, v) is colored by the multi-set of pairs of colors on paths of length two from u to v.

As an example, consider the path of length $n - 1$. Applying 1-dim W-L, the vertices of distance d from an endpoint receive their unique color during step d. The algorithm stops when every vertex "knows" its distance from its closer endpoint. Obviously, this requires $\Theta(n)$ iterations. Using 2-dim W-L, distances up to 2^s are measured in s steps. After only $\log n$ steps, the color of (u, u) (which may be interpreted as the color of vertex u) determines the distance of u from the closer endpoint.

This and other examples suggest, that for $k > 1$, k-dim W-L might always run in just $O(\log n)$ rounds for graphs of size n. In particular, it is very suggestive to make this conjecture for $k = 2$, because this case allows an algebraic treatment. Indeed, it has initiated a vast development in algebra (cellular algebras [11,4] and coherent configurations [7]). It is easy to see that 2-dim W-L corresponds to squaring a matrix A of indeterminates and replacing identical expressions by the same new indeterminate (starting with a modified adjacency matrix where 3 different indeterminates are used for edges, non-edges and diagonal elements).

Assume, instead of this special "squaring" operation, one would do a sequence of corresponding "multiplications" by A. As there can be at most n^2 colors of vertex pairs, this process would stop at the latest with A^{n^2-2}. All higher "powers" would be equal to this one. As a result of this reasoning, one might jump to the conclusion that $O(\log n)$ squaring operations were always sufficient. We will show in this paper that this is not at all the case. This somewhat counterintuitive result is possible, because the just described matrix "product" is not associative.

Section 2 reviews some background information on the basic techniques connecting Weisfeiler-Lehman refinement to logic and games. Section 3 presents the examples for which upper and lower bounds will be proved in Section 5. A simplified view of the pebble games is discussed in Section 4.

2 The Cai-Fürer-Immerman Method

The strength of k-dim W-L has long been an open problem. It has been difficult to find graphs, for which k-dim W-L does not succeed immediately. Already 1-dim W-L identifies random graphs in linear time [1]. For regular graphs, 1-dim W-L cannot even get started. But 2-dim W-L is strong enough to identify shortest cycles and classify the vertices by their distance from the set of vertices covered by shortest cycles. Refining this classification is likely to identify random regular graphs [10] in linear time. It seemed reasonable to conjecture that $f(k)$-dim W-L could identify all degree k graphs for some slow growing function f, e.g., $f(k) = k$. Cai, Fürer, and Immerman [2] have shown that this is very far

from the truth. Indeed $k = \Omega(n)$ is required for graphs of degree 3. We use a modification of their counter-examples to produce graphs which can be identified by k-dim W-L, but only after a linear number of iterations.

Cai, Fürer, and Immerman [2] exhibit an intimate connection between three different approaches to the graph isomorphism problem. These approaches are based on Weisfeiler-Lehman refinement, descriptional complexity, and a version of Ehrenfeucht-Fraïssé games [3,5].

To understand the present paper, it is required to know many definitions and techniques from the Cai, Fürer, and Immerman [2] paper. We start by reviewing some of these notions and their applications.

2.1 Logic Background

Definition 1. For a given language $\mathcal{L}$, the graphs G and H are *$\mathcal{L}$-equivalent* ($G \equiv_{\mathcal{L}} H$) iff the same sentences of $\mathcal{L}$ hold for G and H. Formally, this is expressed as

$$G \models \varphi \quad \Leftrightarrow \quad H \models \varphi \,.$$

for all sentences $\varphi \in \mathcal{L}$.

We say that $\mathcal{L}$ *identifies* the graph G, if $G \equiv_{\mathcal{L}} H$ implies G and H are isomorphic.

We define $\mathcal{L}_k$ to be the set of first-order formulas φ, such that the variables in φ are a subset of $x_1, x_2, \ldots, x_k$. To see the full power of $\mathcal{L}_k$, one has to reuse the same variable many times for different purposes in the same formula — a practice that is not very common in everyday mathematics.

For example, consider the following sentence in $\mathcal{L}_2$.

$$\psi \quad \equiv \quad \forall x_1 \exists x_2 \Big(E(x_1, x_2) \wedge \exists x_1 \big(\neg E(x_1, x_2) \big) \Big)$$

The sentence, ψ, says that every vertex is adjacent to some vertex which is itself not adjacent to every vertex. Note that the first quantifier ($\forall x_1$) refers only to the free occurrence of x_1 within its scope.

The language $\mathcal{L}_k$ is weak in expressing quantitative properties. For example, it is impossible to say that there are k vertices of degree k. On the other hand, it is possible to say that three are $k-3$ vertices of degree 2, even though it has to be formulated somewhat cumbersome.

The language $\mathcal{C}_k$ is a natural extension of $\mathcal{L}_k$, enabling such statements or making them more elegant. For every positive integer i, $\mathcal{C}_k$ allows a quantifier $(\exists i\, x)$ with a straightforward meaning. For example, $(\exists 3\, x)\varphi(x)$ means that there are at least 3 distinct vertices with property φ.

As an example, the following formula in $\mathcal{C}_2$ says that x_i is adjacent to at least two vertices of degree 7.

$$(\exists 2\, x_2)(E(x_1, x_2) \wedge (\exists 7\, x_1) E(x_1, x_2))$$

2.2 Pebbling Games

Let G and H be two graphs, and let m and k be natural numbers. Define the m-move $\mathcal{L}_k$ *game* on G and H as follows. There are two players, and for each variable x_i $(i = 1, \ldots, k)$, there is a pair of pebbles labeled x_i. Initially, the pebbles lie outside the game board containing the graph.

In each move, Player I starts by selecting an $i \in \{1, \ldots, k\}$ and picking up the pair of x_i pebbles. Then he places one of them on a vertex in one of the graphs. Player I is free to select pebbles that have or have not already been placed on the board. Player II must then place the other x_i pebble on a vertex of the other graph.

To define win or loss, consider the subgraphs G' and H' of G and H induced by the pebbled vertices. The pebble respecting mapping f (if it exists) assigns the vertex of G' pebbled by x_i to the vertex of H' pebbled by x_i. Player II loses, if after some move, f does not exist or is not an isomorphism of G' and H'. Player I loses, if Player II plays m moves without losing. Player II has a winning strategy for the $\mathcal{L}_k$ game (without restriction) on the number of moves) if she can play indefinitely without losing against any strategy of Player I.

Some authors call Player II the duplicator, because she wants the two graphs to look the same. They call Player I the spoiler, as he tries to interfere with this goal.

Theorem 1. *[9] Player II has a winning strategy for the $\mathcal{L}_k$ game on G, H iff $G \equiv_{\mathcal{L}_k} H$.*

A modification of the $\mathcal{L}_k$ games provides a combinatorial tool for analyzing the expressive power of $\mathcal{C}_k$. The game board looks the same, and inning is defined as for $\mathcal{L}_k$. Just as in the $\mathcal{L}_k$ game, the two players use k pairs of pebbles. The difference is that each move now has two parts.

- Player I picks up the x_i pebble pair for some i and selects a set A of vertices from one of the graphs. Player II answers with a set B of vertices from the other graph such that $|B| = |A|$.
- Player I places one of the x_i pebbles on some vertex $v \in B$. Player II answers by placing the other x_i pebble on some $u \in A$.

We interpret the first part of a move as an assertion of Player I that there exist $|A|$ vertices in G with a certain property. Player II answers with the same number of such vertices in H. Player I challenges one of the vertices in B and Player II replies with an equivalent vertex from A. Note that it is never an advantage for Player I to include vertices with obviously different properties in A. Again, games and logic are just two sides of the same coin.

Theorem 2. *[8] Player II has a winning strategy for the $\mathcal{C}_k$ game on G, H if and only if $G \equiv_{\mathcal{C}_k} H$.*

2.3 Weisfeiler-Lehman Refinement

One-dimensional Weisfeiler-Lehman refinement (1-dim W-L) is just vertex classification, first by the degree and then by the multi-set of colors of the neighbors, until no color class is split anymore.

For $k > 1$, k-dim W-L is defined as follows. Let G be a graph and let $u = (u_1, \ldots, u_k)$ be a k-tuple of vertices of G. The initial color $W^0(u)$ is defined according to the isomorphism type of u. That is, $W^0(u) = W^0(v)$ iff

$$\forall i \, \forall j \, ((u_i, u_j) \in E \Longleftrightarrow (v_i, v_j) \in E)$$

For each vertex w, we define

$$\begin{aligned}\mathrm{sift}_t(u, w) = \langle & W^t(w, u_2, u_3, \ldots, u_{k-1}, u_k), W^t(u_1, w, u_3, \ldots, u_{k-1}, u_k), \ldots \\ & \ldots, W^t(u_1, u_2, u_3, \ldots, w, u_k), W^t(u_1, u_2, u_3, \ldots, u_{k-1}, w)\rangle\end{aligned}$$

Thus $\mathrm{sift}_t(u, v)$ is the k-tuple of W^t-colors of the k-tuples of vertices obtained by substituting vertex w in turn for each of the k occurrences of a vertex in the k-tuple u.

At time $t+1$, the new colors $W^{t+1}(u)$ and $W^{t+1}(v)$ are the same, if $W^t(u) = W^t(v)$ and the number of w's for which $\mathrm{sift}_t(u, w)$ has any specific value is the same as the number of w's for which $\mathrm{sift}_t(v, w)$ has that same value.

Finally $\overline{W}(u)$ is the stable color of u. It is obtained after at most n^k iterations, i.e., $\overline{W}(u) = W^{n^k}(u)$.

Building on previous work [9,8] the following result has shown the close connection between logic, games, and Weisfeiler-Lehman refinement. Here, the formulas are allowed to have free variables, which are interpreted by the k-tuples u and v respectively.

Theorem 3. *[2] Let G, H be a pair of colored graphs and let (u, v) be a k-configuration on G, H, where $k \geq 1$. Then the following are equivalent:*

1. *$W^m(u) = W^m(v)$ for k-dim W-L*
2. *$G, u \equiv_{\mathcal{C}_{k+1,m}} H, v$*
3. *Player II has a winning strategy for the m-move $\mathcal{C}_{k+1}$ game on (G, H), whose initial configuration is (u, v).*

3 An Example Where k-Dim W-L Is Slow

Our construction of counter-examples starts with a graph G_n^h (see Figure 1), which we call the global graph. We modify G_n^h to obtain 2 graphs $X(G_n^h)$ and $\tilde{X}(G_n^h)$ ("X twist of G_n^h") which are difficult to distinguish by k-dim W-L. For the purpose of forcing k to be big, the global graph has been chosen as an expander [2]. For this paper, we choose the pretty simple grid graph G_n^h.

Now we describe how to modify G_n^h to obtain $X(G_n^h)$. Every vertex of degree d of G_n^h is replaced by 2^{d-1} vertices, which we want to view as the four corners

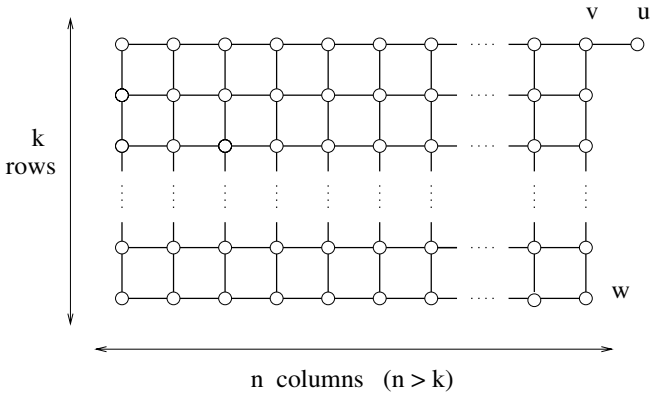


Fig. 1. The global graph G_n^h with $kn + 1$ vertices, where $k \geq 1$ is a constant

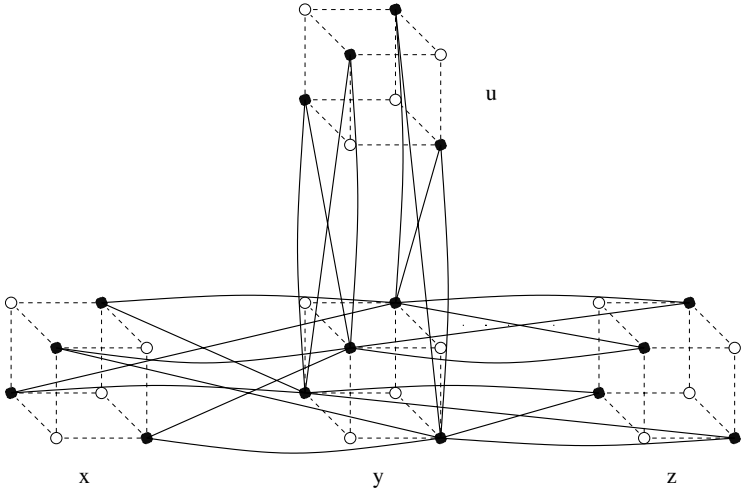


Fig. 2. This figure shows a meta-vertex y and its 3 neighbors u, x, and z. All 4 meta-vertices correspond to vertices of degree 3 in G_n^h. They are therefore represented by 3-dimensional half-cubes. For each meta-vertex, only the 4 dark points are vertices. The 4 white points and the dashed lines are just there to illustrate the cubes. Note the 3 different types of connections of y to its neighbors. The connections to u are left to left and right to right. The connections to x are front to front and back to back. The connections to z are top to top and bottom to bottom. A top to left and bottom to right connection would also be fine, as long as every vertex of degree 3 is represented by a meta-vertex whose connections represent the 3 basic partitions: left-right, top-bottom, and front-back.

of a d-dimensional cube with an even number of coordinates being 1. We refer to these vertices as a half-cube or meta-vertex (see Figure 2).

We might denote the vertices of the half-cube at a vertex v of G_n^h by $v(0,0,0)$, $v(0,1,1), v(1,0,1), v(1,1,0)$. If two vertices u and v of G_n^h are adjacent, then their half-cubes are connected as follows. Say, u is of degree 4 and and $\{u,v\}$ is the third edge of u, and v is of degree 3 and $\{u,v\}$ is the first edge of v. Then, for all $i_1, i_2, i_4, j_2, j_3, \ell \in \{0,1\}$, the vertex $u(i_1, i_2, \ell, i_4)$ is adjacent to $v(\ell, j_2, j_3)$ (provided these vertices exist, i.e., the sum of their coordinates is even).

$\tilde{X}(G_n^h)$ is constructed almost exactly as $X(G_n^h)$ with one exception. We say one edge of G_n^h is twisted.

Definition 2. *To* twist an edge $\{u,v\}$ *of the global graph* G_n^h *means to replace every edge between the meta-vertex* u *and the meta-vertex* v *by a non-edge, and every non-edge by an edge.*

It is not difficult to see that $\tilde{X}(G_n^h)$ and $X(G_n^h)$ are not isomorphic. We cannot make the twist disappear, but we can move it around to any edge of the connected global graph G_n^h. For example, mapping $u(i_1, i_2, i_3, i_4)$ to $u(1-i_1, i_2, i_3, 1-i_4)$ moves a twist from a the first edge of u to its fourth edge.

4 The Global Game

The graphs $X(G_n^h)$ and $\tilde{X}(G_n^h)$ are nicely structured. Nevertheless it is somewhat complicated to analyze the games played on them. Therefore, we investigate a simpler game $\mathcal{G}_k$ that can still adequately describe the original game $\mathcal{C}_k$. The new game is played on the global graph G_n^h rather than the pair $(X(G_n^h), \tilde{X}(G_n^h))$. We therefore call it the global game.

The moves of Player I are very much the same as before. He picks up one of his k pebbles and puts it on a vertex of G_n^h. The moves of Player II are of a very different kind. To describe them, we introduce the following notion of connected components of edges in G_n^h.

Definition 3. *The edges* e, e' *are* connected *if there is a path* $v_0, v_1, \ldots, v_{\ell-1}, v_\ell$ *in* G_n^h *with* $e = (v_0, v_1)$, $e' = (v_{\ell-1}, v_\ell)$, *and none of the interior vertices* $v_1, v_2, \ldots, v_{\ell-1}$ *is holding a pebble.*

We just use the term *component* when we mean a connected component of edges.

A move of Player II just consists of declaring certain components as twisted. The game $\mathcal{G}_k$ starts with no pebbles on the board, and the only component being twisted. At any time, the number of twisted components is odd. When Player I picks up a pebble from the board, two or more components might merge into one component. The new component is twisted iff the number of merged twisted components was odd. When Player I places a pebble on the board, then one component might be replaced by two or more newly formed components. Player II declares the new components as twisted or straight, with the only restriction that the parity of the number of twisted components does not change. When a move of Player I does not split any component, then the answer of Player II consists of doing nothing.

Definition 4. *If a twisted component has size 1, then we say the twist is* trapped.

Player II loses the global game $\mathcal{G}_k$ as soon as any twist is trapped. Player II wins the m-move game, if she can do m moves without losing.

Intuitively, the original game $\mathcal{C}_k$ and the new global game $\mathcal{G}_k$ are equivalent, because of the following reasoning.

- Player I does not really have a reason to place a pebble on any node other than the origin $u(0,\dots,0)$ of a meta-vertex u. So we might just view him as placing the pebble on the meta-vertex (or the corresponding vertex of the global graph G_n^h).
- Unless risking inevitable defeat, Player II better place her pebble on the corresponding meta-vertex. Thus, no selection of a meta-vertex has to be done by Player II. She just selects among the vertices of the given meta-vertex. She does this by selecting twists to move her choice on u into the origin of u.
- Here, we only consider graphs G_n^h without any non-trivial automorphisms. Furthermore, every vertex can easily be identified. Therefore, the global game can be played $\mathcal{L}$-like rather than $\mathcal{C}$-like. No player makes any claims about the existence of more than one vertex with certain properties.

In summary, we are not claiming that every play of the original game $\mathcal{C}_k$ could be simulated by the global game $\mathcal{G}_k$, but we will show that it is of no significant disadvantage for a player to play in a way that can be so simulated.

Definition 5. *Player II plays* proper *in the game $\mathcal{C}_k$, if after any of her moves, it is possible to apply an odd number of twists to $\tilde{X}(G_n^h)$ such that there is a pebble respecting isomorphism between $X(G_n^h)$ and the modified graph $\tilde{X}(G_n^h)$.*

In particular, if Player II plays proper, then she answers every move by a move in the corresponding meta-vertex. Likewise, she answers any set of potential moves by a set of potential moves in corresponding meta-vertices. She is further restricted in placing a pebble within a meta-vertex, should Player I place more than one pebble on the same meta-vertex.

Our graphs G_n^h have the property that there is a unique vertex u of degree 1, distinguishing its neighbor v, and the unique vertex w of degree 2 at distance h from u. All the other vertices are characterized by their distances from v and w.

Lemma 1. *Let the number of pebbles be at least 3. If at any time, Player II does not play proper in $\mathcal{C}_k$, then Player I can force a win in $O(\log n)$ additional moves.*

Proof. The unique characterization of the vertices in G_n^h implies that some distance is wrong, whenever Player II selects a non-matching meta-vertex. With 3 pebbles, Player I can easily exhibit the shorter distance in $O(\log n)$ moves by a divide-and-conquer approach. Hereby, Player I might have a need to identify the vertices u or w of G_n^h. As these vertices are (partly) characterized by their degrees, Player I will use the full power of $\mathcal{C}_k$-moves as follows. When Player II

matches a low degree vertex by a high degree vertex, then Player I proposes the set of neighbors of the high degree vertex, and Player II has no appropriate answer.

Assume now that Player II has always played in the correct meta-vertices, but no set of twists can produce a pebble respecting isomorphism. Then it is not hard to see that there has to be an inconsistency within a meta-vertex containing multiple pebbles. E.g., $X(G_n^h)$ might have 2 pebbles in the front of the half-cube, while $\tilde{X}(G_n^h)$ has one of the corresponding pebbles in the front and one in the back. By selecting in that neighboring meta-vertex which distinguishes front from back, Player I wins in one move.

□

As it does not pay off for Player II to play improper, we can focus now on the case where Player II always plays proper.

Theorem 4. *Assume Player II is restricted to play proper in the game $\mathcal{C}_k$ on the pair $(X(G_n^h), \tilde{X}(G_n^h))$. Then a player has a strategy to win the m-move $\mathcal{C}_k$ game on the pair $(X(G_n^h), \tilde{X}(G_n^h))$ if and only if that player has a strategy to win the m-move global $\mathcal{G}_k$-game.*

Proof. We have to prove four parts.

(a) Player I wins the m-move $\mathcal{C}_k$ game on the pair $(X(G_n^h), \tilde{X}(G_n^h))$. In the simulating global game $\mathcal{G}_k$, Player I has only to be specific about the selection of meta-vertices, but not about his choice within any meta-vertex, while Player II still shows her complete selection. Thus Player I can follow his old winning strategy. When Player I wins the simulated game, some pair of pebble is adjacent in one copy, but not adjacent in the other one. These two pairs correspond to a trapped twist in G_n^h, indicating a win in the simulating game too.

(b) Player I wins the m-move $\mathcal{G}_k$ game on the pair $(X(G_n^h), \tilde{X}(G_n^h))$. In the simulating game $\mathcal{C}_k$, Player I has to make choices within meta-vertices. He always chooses the origin. A trapped twist in the global game $\mathcal{G}_k$ corresponds to an edge vs. non-edge pair in the simulating game implying a win too.

(c) Player II wins the m-move $\mathcal{C}_k$ game on the pair $(X(G_n^h), \tilde{X}(G_n^h))$. As Player II is restricted to proper plays, there is always a placement of twists onto the edges such that her moves are exactly matching the moves of Player I. The placements of twists on edges determine a unique parity of twists in each component, producing the simulating move of Player II. The simulated move produces no conflict if the simulated move did not.

(d) Player II wins the m-move $\mathcal{G}_k$ game on the pair $(X(G_n^h), \tilde{X}(G_n^h))$. The moves of Player II in the $\mathcal{G}_k$-game really describe her strategy to reply to any move of Player I on the same meta-vertex. Player II just follows this strategy. □

5 Upper and Lower Bounds

Theorem 5. *The number of moves sufficient for Player I to win the game $\mathcal{C}_k$ varies as follows depending on the number of pebbles.*

(a) *Player I has a winning strategy in the $\mathcal{C}_{3h}$ game on the pair $(X(G_n^h), \tilde{X}(G_n^h))$ in $O(\log n)$ moves.*
(b) *Player I has a winning strategy in the $\mathcal{C}_{2h+1}$ game on the pair $(X(G_n^h), \tilde{X}(G_n^h))$ in $O(\sqrt{n})$ moves.*
(c) *Player I has a winning strategy in the $\mathcal{C}_{h+1}$ game on the pair $(X(G_n^h), \tilde{X}(G_n^h))$ in $O(n)$ moves.*

Proof. It is sufficient to consider the corresponding $\mathcal{G}_k$ game. We say that Player I builds a *wall* if he places pebbles on all vertices of a cut disconnection the leftmost column form the rightmost (full) column. For example the vertices of one column of G_n^h form a wall.

(a) Having enough pebbles to produce 3 walls in G_n^h, Player I can employ a divide-and-conquer strategy. The pebbles of one wall have only to be removed when the twist is captured between the other 2 walls.
(b) Player I builds a new wall at distance $\sqrt{n}$ from the previous wall starting in the middle and moving towards the twist. As soon as a wall is built that keeps the twist away from the other one, the old wall is no longer needed and its pebbles can be reused. If the twist is located between two walls, then Player I moves one of them slowly inside using the additional pebble.
(c) Player I builds a wall anywhere (best in the middle). Then move it slowly towards the side containing the twist. □

Note that Player I can win the $\mathcal{G}_k$ game on G_n^h by a particularly simple winning strategy. He can build a wall on the left hand side and move it towards the right hand side, step by step decreasing the size of the component containing the twist. All moves of Player I are independent of the moves of Player II.

Theorem 6. *For $k \leq h$, Player II has a winning strategy in the $\mathcal{C}_k$ game on the pair of graphs $(X(G_n^h), \tilde{X}(G_n^h))$.*

Proof. We may look at the corresponding $\mathcal{G}_k$ game. Even for $k = h$, Player I has just enough pebbles to build a wall, but in the next move he has to break it down again. Player II easily maintains a single twist, always in the largest component. □

Corollary 1. *$h-1$-dim W-L cannot detect a difference between the graphs $X(G_n^h)$ and $\tilde{X}(G_n^h)$.*

Corollary 2. *$X(G_n^h)$ and $\tilde{X}(G_n^h)$ agree on all formulas of $\mathcal{C}_h$.*

Definition 6. *The* size *of a component in G_n^h is the number of empty columns in it. A component is* good *if its size is positive.*

Theorem 7. *For $k \leq 2h$, every winning strategy of Player I in the $\mathcal{G}_k$ game on G_n^h requires at least $\Omega(n)$ moves.*

Proof. Let us start with the trivial observation that in G_n^h there are h vertex-disjoint paths between any pair of distinct good components. Thus there is a wall consisting of at least h pebbled vertices between these components. Thus with at most $2h$ pebbles, there are at any time at most 3 components.

We now want to describe a strategy for Player II, that sufficiently delays a win of Player I. In this strategy, Player II always maintains just a single twist. Assume that one good component C_1 of size s_1 exists, and another good component containing the twist is just split into into good component C_2, C_3 with sizes s_2 and s_3 respectively. Let C_2 be the component between C_1 and C_3. Then Player II puts the twist into C_3 if $s_1 + s_2 \leq s_3$, and otherwise into C_2. The following removal of any pebble by Player I breaks a wall, again producing 2 components with the twist being in a component of size at least s_3.

When two good components are formed after m' moves, the twist is in the larger component of size at least $(n-m')/2$. After m moves, the twist is usually in a component of size at least $(n-m')/2-(m-m') = (n+m')/2-m > n/2-m$. There is an exception for the isolated times, when 3 components exist, in which case $n/2 - m$ is a lower bound on the sum of the sizes of the middle and any outer component. Player II does not lose before the twist is in a bad component (of size 0). Thus the number of moves is at least $n/2 = \Omega(n)$. □

Corollary 3. *For $k \leq 2h$, every winning strategy of Player I in the $\mathcal{C}_k$ game on the pair $(X(G_n^h), \tilde{X}(G_n^h))$ requires at least $\Omega(n)$ moves.* □

Theorem 8. *For $k \leq 2h+1$, every winning strategy of Player I in the $\mathcal{G}_k$ game on G_n^h requires at least $\Omega(\sqrt{n})$ moves.*

Proof. As in the proof of Theorem 7, there are at most 3 good components at any time. When 2 good components are formed for the first time, a good strategy for Player II is to move the twist into the larger one. When 3 good component C_1, C_2, C_3 (with $s_i =$ size of C_i) are formed, she has a choice between say C_2 and C_3 where C_2 is between C_1 and C_3. She chooses C_2 if $s_2 > \sqrt{n}$ and $s_3 < n/2 - k$. (This selection could be slightly better optimized without improving the Theorem.) Consider the integer r defined by

$$r = \min(s_1 + s_2, s_3 + s_2, s_2\sqrt{n})$$

if there are 3 good components, and the twist is in C_2. If there are less than 3 good components, then r is defined to be the size of the larger or only good component. When two components are formed form one, then r gets a value of at least $(n-k)/2$. Once the value of r is less than $n/2-k$, it can never decrease by more than $\sqrt{n}$ in a single move. This can be shown by case analysis, where the only interesting case is going form 2 good components to 3. The $\Omega(n)$ lower bound follows immediately. □

Corollary 4. *For $k \leq 2h+1$, every winning strategy of Player I in the $\mathcal{C}_k$ game on the pair $(X(G_n^h), \tilde{X}(G_n^h))$ requires at least $\Omega(\sqrt{n})$ moves.* □

A recent result of Grohe [6] says that determining whether two graphs are $\mathcal{C}_{k+1}$ equivalent, and thus whether they can be distinguished by k-dimensional Weisfeiler-Lehman refinement, is P-complete. Grohe shows the same result for $\mathcal{L}_{k+1}$ equivalence too. This does not imply, but certainly strongly suggests that k-dimensional Weisfeiler-Lehman refinement is slow. Indeed the method of Grohe could also be used to prove Theorem 7. It seems that such a proof would be much more complicated than the proof given in this paper.

Acknowledgment. I want to thank Luitpold Babel for an email conversation in 1994 on some results that implicitly assumed associativity of the multiplication in coherent algebras. This has caused me to discover the main result of this paper.

References

1. L. Babai and L. Kučera, *Graph canonization in linear average time*, 20th Annual Symposium on Foundations of Computer Science (Long Beach, Ca., USA), IEEE Computer Society Press, October 1979, pp. 39–46.
2. Jin-Yi Cai, Martin Fürer, and Neil Immerman, *An optimal lower bound on the number of variables for graph identification*, Combinatorica **12** (1992), no. 4, 389–410.
3. A. Ehrenfeucht, *An application of games to the completeness problem for formalized theories*, Fund. Math. **49** (1960/1961), 129–141.
4. I. A. Faradžev, M. H. Klin, and M. E. Muzichuk, *Cellular rings and groups of automorphisms of graphs*, Investigations in algebraic theory of combinatorial objects, Kluwer Acad. Publ., Dordrecht, 1994, pp. 1–152.
5. Roland Fraïssé, *Sur quelques classifications des systèmes de relations*, Publ. Sci. Univ. Alger. Sér. A. **1** (1954), 35–182 (1955).
6. Martin Grohe, *Equivalence in finite-variable logics is complete for polynomial time*, Combinatorica **19** (1999), no. 4, 507–532.
7. D. G. Higman, *Coherent configurations. I. Ordinary representation theory*, Geometriae Dedicata **4** (1975), no. 1, 1–32.
8. N. Immerman and E. S. Lander, *Describing graphs: A first-order approach to graph canonization*, Alan L. Selman, Editor, Complexity Theory Retrospective, In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988, Springer-Verlag, 1990, pp. 59–81.
9. Neil Immerman, *Upper and lower bounds for first order expressibility*, Journal of Computer and System Sciences **25** (1982), no. 1, 76–98.
10. L. Kučera, *Canonical labeling of regular graphs in linear average time*, Proceedings of the 28th Annual Symposium on Foundations of Computer Science (Los Angeles, CA) (Ashok K. Chandra, ed.), IEEE Computer Society Press, October 1987, pp. 271–279.
11. Boris Weisfeiler (ed.), *On construction and identification of graphs*, Springer-Verlag, Berlin, 1976, With contributions by A. Lehman, G. M. Adelson-Velsky, V. Arlazarov, I. Faragev, A. Uskov, I. Zuev, M. Rosenfeld and B. Weisfeiler, Lecture Notes in Mathematics, Vol. 558.

On Interactive Proofs with a Laconic Prover
(Extended Abstract)

Oded Goldreich[1,⋆], Salil Vadhan[2,⋆⋆], and Avi Wigderson[3,⋆⋆⋆]

[1] Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel
oded@wisdom.weizmann.ac.il
[2] Division of Engineering & Applied Sciences, Harvard University, Cambridge, MA
salil@eecs.harvard.edu, http://eecs.harvard.edu/~salil
[3] School of Mathematics, Institute for Advanced Study, Princeton, NJ
avi@ias.edu

Abstract. We continue the investigation of interactive proofs with bounded communication, as initiated by Goldreich and Håstad (IPL 1998). Let L be a language that has an interactive proof in which the prover sends few (say b) bits to the verifier. We prove that the complement $\bar{L}$ has a *constant-round* interactive proof of complexity that depends only exponentially on b. This provides the first evidence that for **NP**-complete languages, we cannot expect interactive provers to be much more "laconic" than the standard **NP** proof.
When the proof system is further restricted (*e.g.,* when $b = 1$, or when we have perfect completeness), we get significantly better upper bounds on the complexity of $\bar{L}$.

Keywords: interactive proofs, Arthur-Merlin games, sampling protocols, statistical zero knowledge, game theory

1 Introduction

Interactive proof systems were introduce by Goldwasser, Micali and Rackoff [GMR89] in order to capture the most general way in which one party can *efficiently verify* claims made by another, more powerful party.[1] That is, interactive proof systems are two-party randomized protocols through which a computationally unbounded prover can convince a probabilistic polynomial-time verifier of

⋆ Supported by the MINERVA Foundation.
⋆⋆ Work done while at the Institute for Advanced Study, Princeton, NJ, supported by an NSF Mathematical Sciences Postdoctoral Research Fellowship.
⋆⋆⋆ Partially supported by NSF grants CCR-9987845 and CCR-9987077.

[1] Arthur-Merlin games, introduced by Babai [Bab85], are a special type on interactive proofs in which the verifier is restricted to send the outcome of each coin it tosses. Such proof systems are also called *public coin*, and are known to be as expressive as general interactive proofs [GS89]. We warn that the latter assertion refers to the entire class but not to refined complexity measures such as the number of bits sent by the prover (considered below).

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 334–345, 2001.

the membership of a common input in a predetermined language. Thus, interactive proof systems generalize and contain as a special case the traditional "**NP**-proof systems" (in which verification is deterministic and "non-interactive").

It is well-known that this generalization buys us a lot: The *IP Characterization Theorem* of Lund, Fortnow, Karloff, Nisan and Shamir [LFKN92,Sha92] states that every language in **PSPACE** has an interactive proof system, and it is easy to see that only languages in **PSPACE** have interactive proof systems.

It is well-known that the strong expressive power of interactive proofs is largely due to the presence of interaction. In particular, interactive proofs in which a single message is sent (like in **NP**-proofs) yield a complexity class (known as **MA**) that seems very close to **NP**. It is interesting to explore what happens between these extremes of unbounded interaction and no interaction. That is, *what is the expressive power of interactive proofs that utilize a bounded, but nonzero, amount of interaction?*

Interactive Proofs with Few Messages. The earliest investigations of the above question examined the *message complexity* of interactive proofs, *i.e.,* the number of messages exchanged. (Sometimes, we refer to *rounds*, which are a pair of verifier-prover messages.) The Speedup Theorem of Babai and Moran [BM88] (together with [GS89]) shows that the number of messages in an interactive proof can be always be reduced by a constant factor (provided the number of messages remains at least 2). On the other hand, there is a large gap between constant-round interactive proofs and unrestricted interactive proofs. As mentioned above, all of **PSPACE** has a general interactive proof [LFKN92,Sha92]. In contrast, the class **AM** of problems with constant-round interactive proofs is viewed as being relatively close to **NP**. Specifically, **AM** lies in the second level of the polynomial-time hierarchy [BM88], cannot contain **coNP** unless the polynomial-time hierarchy collapses [BHZ87], and actually equals **NP** under plausible circuit complexity assumptions [AK97,KvM99,MV99].

Laconic Provers. A more refined investigation of the above question was initiated by Goldreich and Håstad [GH98], who gave bounds on the complexity of languages possessing interactive proofs with various restrictions on the *number of bits* of communication and/or randomness used. One of the restrictions they considered, and the main focus of our investigation, limits the number of bits sent from the prover to the verifier by some bound b. That is, what languages can be proven by "laconic" provers?

Since the prover is trying to convey something to the verifier, this seems to be the most interesting direction of communication. Moreover, for applications of interactive proofs (*e.g.,* in cryptographic protocols), it models the common situation in which communication is more expensive in one direction (*e.g.,* if the prover is a handheld wireless device).

On one hand, we know of interactive proofs for several "hard" problems (Quadratic Nonresiduosity [GMR89], Graph Nonisomorphism [GMW91], and others [GK93,GG00,SV97]) in which the communication from the prover to the verifier is severely bounded (in fact, to one bit). On the

other hand, no such proof systems were known for **NP**-complete problems, nor was there any indication of impossibility (except when additional constraints are imposed [GH98]). In this work, we provide strong evidence of impossibility.

Our Results. Consider interactive proofs in which the prover sends at most $b = b(n)$ bits to the verifier on inputs of length n. Goldreich and Håstad [GH98, Thm. 4] placed such languages in $\mathbf{BPTIME}^{\mathbf{NP}}(T)$, where $T = \mathrm{poly}(n) \cdot 2^{\mathrm{poly}(b)}$, which clearly implies nothing for languages in **NP**. In contrast, we show that the *complements* of such languages have *constant-round* interactive proofs of complexity T (*i.e.,* the verifier's computation time and the total communication is bounded by T). In particular, **NP**-complete problems cannot have interactive proofs in which in which the prover sends at most polylogarithmically many bits to the verifier unless **coNP** is in the quasipolynomial analogue of **AM**. In fact, assuming **NP** has constant-round interactive proofs with logarithmic prover-to-verifier communication we conclude $\mathbf{coNP} \subset \mathbf{AM}$. As mentioned above, this is highly unlikely.

We obtain stronger results in two special cases:

1. We show that if a language has an interactive proof of perfect completeness (*i.e.,* , zero error probability on YES instances) in which the prover sends at most $b(n)$ bits, then it is in $\mathbf{coNTIME}(T)$, where $T(n) = 2^{b(n)} \cdot \mathrm{poly}(n)$. Thus, unless $\mathbf{NP} = \mathbf{coNP}$, NP-complete languages cannot have interactive proof systems of perfect completeness in which the prover sends at most logarithmically many bits.
2. We show that if a language has an interactive proof in which the prover sends a single bit (with some restrictions on the error probabilities), then it has a statistical zero-knowledge interactive proof; that is, is in the class **SZK**. This is a stronger conclusion than our main result because $\mathbf{SZK} \subseteq \mathbf{AM} \cap \mathbf{coAM}$, as shown by Fortnow [For89] and Aiello and Håstad [AH91]. Recalling that Sahai and Vadhan [SV97] showed that any language in **SZK** has an interactive proof in which the prover sends a single bit, we obtain a surprising equivalence between these two classes.[2]

Lastly, we mention one easy, but apparently new, observation regarding message complexity. A question that is left open by the results mentioned earlier is what happens "in between" constant rounds and polynomially many rounds. Phrased differently, can the Speedup Theorem of Babai and Moran be improved to show that $m(n)$-message interactive proofs are no more powerful than $m'(n)$-message ones for some $m' = o(m)$? By combining careful parameterizations of [LFKN92,BM88], we observe that such an improvement is unlikely. More precisely, for every nice function m, we show that there is a language which has an $m(n)$-message interactive proof but not an $o(m(n))$-message one, provided that #SAT is not contained in the subexponential analogue of **coAM**.

[2] In addition, if the error probabilities are sufficiently small, we also are able to reduce interactive proofs in which the prover sends a single *message* of several bits (*e.g.,* $O(\mathrm{loglog}\, n)$ bits) to the 1-bit case above. But we omit these results from this extended abstract due to space constraints.

Additional Related Work. It should be noted that the results of Goldreich and Håstad are significantly stronger when further restrictions are imposed in addition to making the prover laconic. In particular, they obtain an upper bound of $\mathbf{BPTIME}(T)$ (rather than $\mathbf{BPTIME}^{\mathbf{NP}}(T)$), with $T = 2^{\mathrm{poly}(b)} \cdot \mathrm{poly}(n)$ for languages possessing either of the following kinds of interactive proofs: (a) *public-coin* proofs in which the prover sends at most b bits, (b) proofs in which the communication *in both directions* is bounded by b.

There has also been a body of research on the expressive power of *multi-prover interactive proofs* (MIP's) and *probabilistically checkable proofs* (PCP's) with low communication, because of the importance of the communication parameter in their applications to inapproximability. In particular, Bellare, Goldreich, and Sudan [BGS98] give negative results about the expressive power of "laconic" PCP's and MIP's. One-query probabilistically checkable proofs are equivalent to interactive proofs in which the prover sends a single message, so our results provide bounds on the former.

Our work is also related to work on *knowledge complexity*. Knowledge complexity, proposed by [GMR89], aims to measure how much "knowledge" is leaked from the prover to the verifier in an interactive proof. Several measures of knowledge complexity were proposed by Goldreich and Petrank [GP99], and series of works provided upper bounds on the complexity of languages having interactive proofs with low knowledge complexity [GP99,GOP98,PT96,SV97]. These results are related to, but incomparable to ours.

For example, Petrank and Tardos [PT96] showed that languages having knowledge complexity $k = O(\log n)$ are contained in $\mathbf{AM} \cap \mathbf{coAM}$. While it is true that the knowledge complexity of an interactive proof is bounded by the amount of prover-to-verifier communication, their result does not yield anything interesting for laconic interactive proofs. The reason is that their result only applies to interactive proofs with error probabilities significantly smaller than 2^{-k}, and it is easy to see that interactive proofs with prover-to-verifier communication $k = O(\log n)$ error probability $\ll 2^{-k}$ only capture **BPP** (and hence are uninteresting). Our results apply even for constant error probabilities.

Sahai and Vadhan [SV97] (improving [GP99]) showed that languages with logarithmic knowledge complexity in the "hint sense" collapse to **SZK**, and their result applies even if the error probabilities are constant. However, this is also incomparable to ours, for the "hint sense" is the one measure of knowledge complexity which is *not* bounded by the prover-to-verifier communication.

Finally, it is important to note that the situation is dramatically different for *argument systems* [BCC88] (also known as *computationally sound proofs*). These are like interactive proofs, but the soundness condition is restricted to polynomial-time provers. Kilian [Kil92] showed that **NP** has laconic argument systems if strong collision-resistant hash functions exist. Specifically, under a strong enough (but still plausible) assumption, **NP** has *public-coin* arguments in which the verifier's randomness and the communication in both directions is *polylogarithmic*. Combined with [GH98], this provides a strong separation between the efficiency of arguments versus interactive proofs for **NP**; and our

results extend this separation to the case that only the prover-to-verifier communication is counted (and the interactive proof is not required to be public coin).

2 Preliminaries

We assume that the reader is familiar with the basic concepts underlying interactive proofs (and public-coin interactive proofs) (see *e.g.,* [Sip97,Gol99,Vad00]). Throughout, we work with interactive proofs for *promise problems* rather than languages. More precisely, a promise problem $\Pi = (\Pi_Y, \Pi_N)$ is a pair of disjoint sets of strings, corresponding to YES and NO instances, respectively. In other words, a promise problem is simply a decision problem in which some inputs are excluded. The definition of interactive proofs is extended to promise problems in the natural way: we require that when the input is a YES instance, the prover convinces the verifier to accept with high probability (*completeness*); and when the input is a NO instance, the verifier accepts with low probability no matter what strategy the prover follows (*soundness*). Working with promise problems rather than languages only makes our results stronger (except for one direction of Theorem 4.4).

We denote by $\mathbf{IP}(b, m)$ (resp., $\mathbf{AM}(b, m)$) the class of problems having interactive proofs (resp., public-coin interactive proofs) in which the prover sends a total of at most b bits, and the total number of messages exchanged (in both directions) is at most m. Note that b and m are integer functions of the common input length, denoted n. When b is not polynomial in n, it will be understood that we talk of a generalization in which the verifier is allowed time polynomial in b and n (rather than just in n). Unless specified differently, we refer to proof systems with completeness probability 2/3 and soundness probability 1/3.

We denote $\mathbf{IP}(b) = \mathbf{IP}(b, 2b)$; that is, making only the trivial bound on the number of messages exchanged. We denote by $\mathbf{IP}^+$ the analogue of $\mathbf{IP}$ when the proof system has perfect completeness (*i.e.,* completeness probability 1). The class of problems with constant-round interactive proofs is denoted $\mathbf{AM} \stackrel{\text{def}}{=} \mathbf{AM}(\text{poly}(n), 2) = \mathbf{IP}(\text{poly}(n), O(1))$. (The second equality is by Thms 2.3 and 2.4 below.) When we wish to specify the completeness probability $c = c(n)$ and soundness probability $s = s(n)$ we will use subscripts: $\mathbf{IP}_{c,s}$ and $\mathbf{AM}_{c,s}$.

Using the above notations, we recall the main results of Goldreich and Håstad, which are the starting point for our work.

Theorem 2.1 ([GH98]). $\mathbf{AM}(b, m) \subseteq \mathbf{BPTIME}(\text{poly}(2^b, m^m, n))$

Theorem 2.2 ([GH98]). $\mathbf{IP}(b, m) \subseteq \mathbf{BPTIME}(\text{poly}(2^b, m^m, n))^{\mathbf{NP}}$

We also state some standard results that we will use:

Theorem 2.3 ([BM88]). $\mathbf{AM}(b, m) \subseteq \mathbf{AM}(b^2 \cdot \text{poly}(m), \lceil m/2 \rceil) \subseteq \mathbf{AM}((b \cdot m)^{O(m)}, 2)$.

Theorem 2.4 ([GS89]). $\mathbf{IP}(b, m) \subseteq \mathbf{AM}(\text{poly}(b, n), m)$.

Theorem 2.5 ([BHZ87]). *If* $\mathbf{coNP} \subseteq \mathbf{AM}(b,2)$*, then* $\mathbf{\Sigma_2} \subseteq \mathbf{\Pi_2}(\mathrm{poly}(n,b))$*. In particular, if* $\mathbf{coNP} \subset \mathbf{AM}$*, then the polynomial-time hierarchy collapses to* $\mathbf{PH} = \mathbf{\Sigma_2} = \mathbf{\Pi_2}$*.*

Above and throughout the paper, $\mathbf{\Sigma_i}(t(n))$ (resp., $\mathbf{\Pi_i}(t(n))$) denotes the class of problems accepted by $t(n)$-time alternating Turing machines with i alternations beginning with an existential (resp., universal) quantifier. Thus, $\mathbf{\Sigma_i} \stackrel{\text{def}}{=} \mathbf{\Sigma_i}(\mathrm{poly}(n))$ and $\mathbf{\Pi_i} \stackrel{\text{def}}{=} \mathbf{\Pi_i}(\mathrm{poly}(n))$ comprise the i'th level of the polynomial-time hierarchy.

We will also consider **SZK**, the class of problems possessing statistical zero-knowledge interactive proofs. Rather than review the definition here, we will instead use a recent characterization of it in terms of complete problems which will suffice for our purposes. For distributions X and Y, let $\Delta(X,Y)$ denote their *statistical difference* (or *variation distance*, *i.e.*, $\Delta(X,Y) = \max_S |\Pr[X \in S] - \Pr[Y \in S]|$. We will consider distributions specified by circuits which sample from them. More precisely, a circuit with m input gates and n output gates can be viewed as a sampling algorithm for the distribution on $\{0,1\}^n$ induced by evaluating the circuit on m random input bits. Statistical Difference is the promise problem $\mathrm{SD} = (\mathrm{SD}_Y, \mathrm{SD}_N)$, where

$$\mathrm{SD}_Y = \{(X,Y) \ : \ \Delta(X,Y) \geq 2/3\}$$
$$\mathrm{SD}_N = \{(X,Y) \ : \ \Delta(X,Y) \leq 1/3\},$$

where X and Y are probability distributions specified by circuits which sample from them. More generally, for any $1 \geq \alpha > \beta \geq 0$, we will consider variants $\mathrm{SD}^{\alpha,\beta}$, where the thresholds of $2/3$ and $1/3$ are replaced with α and β respectively.

Theorem 2.6 ([SV97]). *For any constants* $1 > \alpha^2 > \beta > 0$, $\mathrm{SD}^{\alpha,\beta}$ *is complete for* **SZK**.

The following results about **SZK** are also relevant to us.

Theorem 2.7 ([For89,AH91]). $\mathbf{SZK} \subseteq \mathbf{AM} \cap \mathbf{coAM}$.

Theorem 2.8 ([Oka00]). **SZK** *is closed under complement.*

Theorem 2.9 ([SV97]). $\mathbf{SZK} \subseteq \mathbf{IP}_{1-2^{-n},1/2}(1)$.

3 Formal Statement of Results

We improve over Theorem 2.2, and address most of the open problems suggested in [GH98, Sec. 3]. Our main results are listed below.

For one bit of prover-to-verifier communication, we obtain a collapse to **SZK**.

Theorem 3.1. *For every pair of constants* c, s *such that* $1 > c^2 > s > c/2 > 0$, $\mathbf{IP}_{c,s}(1) = \mathbf{SZK}$.

With Theorem 2.8, this gives:

Corollary 3.2. *For every c, s as in Thm. 3.1,* $\mathbf{IP}_{c,s}(1)$ *is closed under complement.*

For more rounds of communication, we first obtain the following result for interactive proofs with perfect completeness (denoted by $\mathbf{IP}^+$):

Theorem 3.3. $\mathbf{IP}^+(b) \subseteq \mathbf{coNTIME}(2^b \cdot \mathrm{poly}(n))$. *In particular,* $\mathbf{IP}^+(O(\log n)) \subseteq \mathbf{coNP}$.

In the general case (*i.e.,* with imperfect completeness), we prove:

Theorem 3.4. $\mathbf{IP}(b, m) \subseteq \mathbf{coAM}(2^b \cdot \mathrm{poly}(m^m, n), O(m))$. *In particular,* $\mathbf{IP}(O(\log n), m) \subseteq \mathbf{coAM}(\mathrm{poly}(n), O(m))$, *for* $m = O(\log n / \log\log n)$,

The above theorems provide first evidence that **NP**-complete problems cannot have interactive proof systems in which the prover sends very few bits. Further evidence toward this claim is obtained by applying Theorems 2.3 and 2.5:

Corollary 3.5. $\mathbf{IP}(b, m) \subseteq \mathbf{coAM}(\mathrm{poly}(2^b, m^m, n)^m, 2)$. *In particular,* $\mathbf{IP}(O(\log n), O(1)) \subset \mathbf{coAM}$ *and* $\mathbf{IP}(\mathrm{polylog}\, n) \subset \widetilde{\mathbf{coAM}}$.

Corollary 3.6. $\mathbf{NP} \not\subseteq \mathbf{IP}(O(\log n), O(1))$ *unless the polynomial-time hierarchy collapses (to* $\mathbf{\Sigma_2} = \mathbf{\Pi_2}$*).* $\mathbf{NP} \not\subseteq \mathbf{IP}(\mathrm{polylog}\, n)$ *unless* $\mathbf{\Sigma_2} \subseteq \widetilde{\mathbf{\Pi}}_2$.

Above, $\widetilde{\mathbf{coAM}}$ and $\widetilde{\mathbf{\Pi}}_2$ denote the quasipolynomial-time ($2^{\mathrm{polylog}\, n}$) analogues of **coAM** and $\mathbf{\Pi_2}$.

Finally, we state our result on message complexity.

Theorem 3.7. *Let* $m(n) \leq n/\log n$ *be any "nice" growing function. Then* $\mathbf{AM}(\mathrm{poly}(n), m(n)) \neq \mathbf{AM}(\mathrm{poly}(n), o(m(n)))$ *unless* $\#\mathrm{SAT} \in \mathbf{AM}(2^{o(n)}, 2)$.

Note that, by Theorem 2.4, it is irrelevant whether we use **IP** or **AM** in this theorem.

Due to space constraints, we only present proofs of Theorems 3.1 and 3.3 in this extended abstract. The proof of our main result (Theorem 3.4) is significantly more involved, and will be given in the full version of the paper.

4 Extremely Laconic Provers (Saying Only One Bit)

In this section, we prove Theorem 3.1. The proof is based on the following lemma, along with previous results.

Lemma 4.1. *Every problem in* $\mathbf{IP}_{c,s}(1)$ *reduces to* $\mathrm{SD}^{c,s}$.

Proof. Let (P, V) be an interactive proof for some problem so that the prover sends a single bit during the entire interaction. We may thus assume that on input x and internal coin tosses r, the verifier first sends a message $y = V_x(r)$, the prover answers with a bit $\sigma \in \{0, 1\}$, and the verifier decides whether to accept or reject by evaluating the predicate $V_x(r, \sigma) \in \{0, 1\}$.

A special case — unique answers. To demonstrate the main idea, we consider first the natural case in which for every pair (x, r) there exists *exactly one* σ such that $V_x(r, \sigma) = 1$. (Note that otherwise, the interaction on input x and verifier's internal coin tosses r is redundant, since the verifier's final decision is unaffected by it.) For this special case (which we refer to as *unique answers*), we will prove the following:

Claim 4.2. *If a problem has an* $\mathbf{IP}_{c,s}(1)$ *proof system with unique answers, then it reduces to* $\mathrm{SD}^{2c-1,2s-1}$.

Let $\sigma_x(r)$ denote the unique σ satisfying $V_x(r, \sigma) = 1$. The prover's ability to convince the verifier is related to the amount of information regarding $\sigma_x(r)$ that is revealed by $V_x(r)$. For example, if for some x, $\sigma_x(r)$ is determined by $V_x(r)$ then the prover can convince the verifier to accept x with probability 1 (by replying with $\sigma_x(r)$). If, on the other hand, for some x, $\sigma_x(r)$ is statistically independent of $V_x(r)$ (and unbiased), then there is no way for the prover to convince the verifier to accept x with probability higher than $1/2$. This suggests the reduction $x \mapsto (C_x^1, C_x^2)$, where $C_x^1(r) \stackrel{\text{def}}{=} (V_x(r), \sigma_x(r))$ and $C_x^2(r) \stackrel{\text{def}}{=} (V_x(r), \overline{\sigma_x}(r))$, where $\overline{b}$ denotes the complement of a bit b.

Now we relate the statistical difference between the distributions sampled by C_x^1 and C_x^2 to the maximum acceptance probability of the verifier. Since the first components of C_x^1 and C_x^2 are distributed identically, their statistical difference is exactly the average over the first component $V_x(r)$ of the statistical difference between the second components conditioned on $V_x(r)$. That is,

$$\Delta(C_x^1, C_x^2) = \mathop{\mathrm{E}}_{y \leftarrow V_x} \left[\Delta\left(\sigma_x|_y, \overline{\sigma_x}|_y\right) \right],$$

where $\sigma_x|_y$ denotes the distribution of $\sigma_x(r)$ when r is uniformly distributed among $\{r' : V_x(r') = y\}$. For any y and $b \in \{0, 1\}$, let $q_{b|y}$ denote the probability that $\sigma_x|_y = b$. Then, for any fixed y, $\Delta\left(\sigma_x|_y, \overline{\sigma_x}|_y\right) = |q_{1|y} - q_{0|y}| = 2q_y - 1$, where $q_y \stackrel{\text{def}}{=} \max_{b \in \{0,1\}}\{q_{b|y}\} \geq \frac{1}{2}$. So, we have:

$$\Delta(C_x^1, C_x^2) = \mathop{\mathrm{E}}_{y \leftarrow V_x} \left[2q_y - 1 \right].$$

On the other hand, the optimal prover strategy in (P, V) is: upon receiving y, respond with b that maximizes $q_{b|y}$. When the prover follows this strategy, we have

$$\Pr[V \text{ accepts } x] = \mathop{\mathrm{E}}_{y \leftarrow V_x} \left[q_y \right].$$

Putting the last two equations together, we conclude that $\Delta(C_x^1, C_x^2) = 2 \cdot \Pr[V \text{ accepts } x] - 1$.[3] Thus if the proof system has completeness and soundness error bounds c and s, respectively, then the reduction maps instances to

[3] Note that under the hypothesis of the special case, for every x the prover may convince the verifier to accept x with probability at least $1/2$ (and so such a non-trivial proof system must have soundness at least $1/2$).

pairs having distance bounds $2c-1$ and $2s-1$, respectively.[4] This establishes Claim 4.2.

The general case. We now proceed to deal with the general case in which there may exist pairs (x,r) so that either both σ's or none of them satisfy $V_x(r,\sigma)=1$. We do so by reducing this general case to the special case.

Claim 4.3. *If a problem is in* $\mathbf{IP}_{c,s}(1)$*, then it has an* $\mathbf{IP}_{(1+c)/2,(1+s)/2}(1)$ *proof system with unique answers.*

Clearly, the lemma follows from this claim and the previous one, so we proceed to prove the claim.

Proof of claim. Let (P,V) be a general $\mathbf{IP}_{c,s}$ proof system. Consider the following modified verifier strategy.

$V'(x)$**:** Generate coin tosses r for the original verifier and do one of the following based on the number j of possible prover responses σ for which $V_x(r,\sigma)=1$.

- $[j=2]$ Send the prover a special message "`respond with 1`" and accept if the prover responds with 1.
- $[j=1]$ Randomly do one of the following (each with prob. 1/2):
  - Send the prover $y=V_x(r)$ and accept if the prover responds with the unique σ such that $V_x(r,\sigma)=1$.
  - Send the prover a special message "`respond with 1`" and accept if the prover responds with 1.
- $[j=0]$ Choose a random bit σ. Send the prover a special message "`guess my bit`" and accept if the prover responds with σ.

Clearly, V' has unique answers. It can be shown that if an optimal prover makes V accept with probability δ, then an optimal prover makes V' accept with probability $(1+\delta)/2$. Claim 4.3 follows. □

■

Theorem 3.1 follows from Lemma 4.1, Theorem 2.6, and Theorem 2.9. Details will be given in the full version of the paper. The $c^2 > s$ constraint in Theorem 3.1 is due to the analogous constraint in Theorem 2.6. Indeed, we can establish the following equivalence (also to be proven the full version of the paper):

Theorem 4.4. *The following are equivalent.*

1. *For every* α,β *such that* $1>\alpha>\beta>0$, $\mathrm{SD}^{\alpha,\beta}$ *is in* **SZK** *(and is therefore also complete).*
2. *For every* c,s *such that* $1>c>s>c/2>0$, $\mathbf{IP}_{c,s}(1)=\mathbf{SZK}$.

Finally, we remark that the condition $s>c/2$ in Theorems 3.1 and 4.4 is necessary, for $\mathbf{IP}_{c,s}(1)=\mathbf{BPP}$ for any $s<c/2$.

[4] Note that this relationship is reversed by the natural $\mathbf{IP}(1)$ system for $\mathrm{SD}^{\alpha,\beta}$ in which the verifier selects at random a single sample from one of the two distributions and asks the prover to guess which of the distributions this sample came from. If the distributions are at distance δ then the prover succeeds with probability $\frac{1}{2}+\frac{\delta}{2}$. Thus applying this proof system to $\mathrm{SD}^{2c-1,2s-1}$ we obtain completeness and soundness bounds c and s, respectively.

5 Laconic Provers with Perfect Completeness

In this section, we prove Theorem 3.3.

Theorem 3.3 (restated): If a problem Π has an interactive proof system with perfect completeness in which the prover-to-verifier communication is at most $b(\cdot)$ bits then $\Pi \in \mathbf{coNTIME}(2^{b(n)} \cdot \mathrm{poly}(n))$.

Proof. We take a slightly unusual look at the interactive proof system for Π, viewing it as a "progressively finite game" between two players P^* and V^*. P^* corresponds to the usual prover strategy and its aim is to make the original verifier accept the common input. V^* is a "cheating verifier" and its aim is to produce an interaction that looks legal and still makes the original verifier reject the common input.

To make this precise, let $b = b(n)$ be the bound on the prover-to-verifier communication in (P, V) on inputs of length n, and let $m = m(n)$ be the number of messages exchanged. Without loss of generality, we may assume that the V sends all its coin tosses in the last message. A *transcript* is a sequence of m strings, corresponding to (possible) messages exchanged between P and V. We call a transcript t *consistent* (for x) if every verifier message in t is the message V would have sent given input x, the previous messages in t, and the coin tosses specified by the last message in t. We call a consistent t *rejecting* if V would reject at the end of such an interaction.

Now, the game between P^*_x and V^*_x has the same structure as the interaction between P and V on input x: a total of m messages are exchanged and P^*_x is allowed to send at most b bits. The game between P^*_x and V^*_x yields a transcript t. We say that V^*_x *wins* if t is consistent and rejecting, and that P^*_x wins otherwise. We stress that V^*_x need not emulate the original verifier nor is it necessarily implemented in probabilistic polynomial time.

This constitutes a "perfect information finite game in extensive form" (also known as a "progressively finite game") and Zermelo's Theorem (*cf.,* [Tuc95, Sec 10.2]) says that exactly one of the two players has a *winning strategy* — that is, a (deterministic) strategy that will guarantee its victory no matter how the other party plays.

Using the perfect completeness condition, we infer that if the common input x is a YES instance then there exists a winning strategy for P^*_x. (This is because the optimal prover for the original interactive proof wins whenever V^*_x plays in a manner consistent with some sequence of coin tosses for the original verifier, and it wins by definition if the V^*_x plays inconsistently with any such sequence.) On the other hand, by the soundness condition, if the common input is a NO instance then there exists no winning strategy for P^*_x. (This is because in this case no prover strategy can convince the original verifier with probability 1.) By the above, it follows that whenever the common input is a NO instance there exists a winning strategy for V^*_x.

Thus, a proof that x is a NO instance consists of a winning strategy for V^*_x. Such strategy is a function mapping partial transcripts of P^*_x messages to

the next V_x^* message. Thus, such a strategy is fully specified by a function from $\cup_{i=0}^{b}\{0,1\}^i$ to $\{0,1\}^{\mathrm{poly}(n)}$, and has description length $\mathrm{poly}(n)\cdot 2^{b(n)+1}$. To verify that such a function constitutes a winning strategy for V_x^*, one merely tries all possible deterministic strategies for the P_x^* (i.e., all possible $b(n)$-bit long strings). The theorem follows. ■

References

[AH91] William Aiello and Johan Håstad. Statistical zero-knowledge languages can be recognized in two rounds. *Journal of Computer and System Sciences*, 42(3):327–345, June 1991.

[AK97] V. Arvind and J. Köbler. On resource-bounded measure and pseudorandomness. In *Proceedings of the 17th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 235–249. LNCS 1346, Springer-Verlag, 1997.

[Bab85] László Babai. Trading group theory for randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 421–429, Providence, Rhode Island, 6–8 May 1985.

[BM88] László Babai and Shlomo Moran. Arthur-Merlin games: A randomized proof system and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36:254–276, 1988.

[BGS98] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, PCPs, and nonapproximability—towards tight results. *SIAM Journal on Computing*, 27(3):804–915 (electronic), 1998.

[BHZ87] Ravi B. Boppana, Johan Håstad, and Stathis Zachos. Does co-NP have short interactive proofs? *Information Processing Letters*, 25:127–132, 1987.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, October 1988.

[For89] Lance Fortnow. The complexity of perfect zero-knowledge. In Silvio Micali, editor, *Advances in Computing Research*, volume 5, pages 327–343. JAC Press, Inc., 1989.

[Gol99] Oded Goldreich. *Modern Cryptography, Probabilistic Proofs, and Pseudorandomness*. Number 17 in Algorithms and Combinatorics. Springer-Verlag, 1999.

[GG00] Oded Goldreich and Shafi Goldwasser. On the limits of nonapproximability of lattice problems. *Journal of Computer and System Sciences*, 60(3):540–563, 2000.

[GH98] Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Information Processing Letters*, 67(4):205–214, 1998.

[GK93] Oded Goldreich and Eyal Kushilevitz. A perfect zero-knowledge proof system for a problem equivalent to the discrete logarithm. *Journal of Cryptology*, 6:97–116, 1993.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.

[GOP98] Oded Goldreich, Rafail Ostrovsky, and Erez Petrank. Computational complexity and knowledge complexity. *SIAM Journal on Computing*, 27(4):1116–1141, August 1998.

[GP99] Oded Goldreich and Erez Petrank. Quantifying knowledge complexity. *Computational Complexity*, 8(1):50–98, 1999.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, February 1989.

[GS89] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In Silvio Micali, editor, *Advances in Computing Research*, volume 5, pages 73–90. JAC Press, Inc., 1989.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on the Theory of Computing*, pages 723–732, Victoria, British Columbia, Canada, 4–6 May 1992.

[KvM99] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, pages 659–667, Atlanta, 1–4 May 1999.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, October 1992.

[MV99] Peter Bro Miltersen and N.V. Vinodchandran. Derandomizing Arthur–Merlin games using hitting sets. In *40th Annual Symposium on Foundations of Computer Science*, New York, NY, 17–19 October 1999. IEEE.

[Oka00] Tatsuaki Okamoto. On relationships between statistical zero-knowledge proofs. *Journal of Computer and System Sciences*, 60(1):47–108, February 2000.

[PT96] Erez Petrank and Gábor Tardos. On the knowledge complexity of $\mathcal{NP}$. In *37th Annual Symposium on Foundations of Computer Science*, pages 494–503, Burlington, Vermont, 14–16 October 1996. IEEE.

[SV97] Amit Sahai and Salil P. Vadhan. A complete promise problem for statistical zero-knowledge. In *38th Annual Symposium on Foundations of Computer Science*, pages 448–457, Miami Beach, Florida, 20–22 October 1997. IEEE.

[Sha92] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, October 1992.

[Sip97] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997.

[Tuc95] Alan Tucker. *Applied combinatorics*. John Wiley & Sons Inc., New York, third edition, 1995.

[Vad00] Salil Vadhan. Probabilistic proof systems I: Interactive and zero-knowledge proofs. Lecture Notes from the *IAS/PCMI Graduate Summer School on Computational Complexity*, August 2000. Available from `http://eecs.harvard.edu/~salil/`.

Quantum Complexities of Ordered Searching, Sorting, and Element Distinctness*

Peter Høyer [1,**], Jan Neerbek [2], and Yaoyun Shi [3]

[1] Dept. of Comp. Sci., University of Calgary, Alberta, Canada T2N 1N4
hoyer@cpsc.ucalgary.ca
[2] Dept. of Comp. Sci., University of Aarhus, DK–8000 Århus C, Denmark
neerbek@daimi.au.dk
[3] Dept. of Comp. Sci., Princeton University, Princeton, NJ 08544, USA
shiyy@cs.princeton.edu

Abstract. We consider the quantum complexities of the following three problems: searching an ordered list, sorting an un-ordered list, and deciding whether the numbers in a list are all distinct. Letting N be the number of elements in the input list, we prove a lower bound of $\frac{1}{\pi}(\ln(N)-1)$ accesses to the list elements for ordered searching, a lower bound of $\Omega(N \log N)$ binary comparisons for sorting, and a lower bound of $\Omega(\sqrt{N} \log N)$ binary comparisons for element distinctness. The previously best known lower bounds are $\frac{1}{12}\log_2(N) - O(1)$ due to Ambainis, $\Omega(N)$, and $\Omega(\sqrt{N})$, respectively. Our proofs are based on a weighted all-pairs inner product argument.

In addition to our lower bound results, we give a quantum algorithm for ordered searching using roughly $0.631 \log_2(N)$ oracle accesses. Our algorithm uses a quantum routine for traversing through a binary search tree faster than classically, and it is of a nature very different from a faster algorithm due to Farhi, Goldstone, Gutmann, and Sipser.

1 Introduction

The speedups of quantum algorithms over classical algorithms have been a main reason for the current interests on quantum computing. One central question regarding the power of quantum computing is: How much speedup is possible? Although dramatic speedups seem possible, as in the case of Shor's [16] algorithms for factoring and for finding discrete logarithms, *provable* speedups are found only in restricted models such as the black box model.

In the black box model, the input is given as a black box, so that the only way the algorithm can obtain information about the input is via queries, and the complexity measure is the number of queries. Many problems that allow provable quantum speedups can be formulated in this model, an example being the unordered search problem considered by Grover [14]. Several tight lower

* Research supported by the EU fifth framework program QAIP, IST-1999-11234, and the National Science Foundation under grant CCR-9820855.

** Research conducted in part while at BRICS, University of Aarhus, Denmark.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 346–357, 2001.

bounds are now known for this model, most of them being based on techniques introduced in [5,3,2].

We study the quantum complexities of the following three problems.

Ordered searching. Given a list of numbers $x = (x_0, x_1, \ldots, x_{N-1})$ in non-decreasing order and some number y, find the minimal i such that $y \leq x_i$. We assume that $x_{N-1} = \infty > y$ so that the problem is always well-defined.

Sorting. Given a list of numbers $x = (x_0, x_1, \ldots, x_{N-1})$, output a permutation σ on the set $\{0, \ldots, N-1\}$ so that the list $(x_{\sigma(0)}, x_{\sigma(1)}, \ldots, x_{\sigma(N-1)})$ is in non-decreasing order.

Element distinctness. Given a list of numbers $x = (x_0, x_1, \ldots, x_{N-1})$, are they all distinct?

These problems are closely related and are among the most fundamental and most studied problems in the theory of algorithms. They can also be formulated naturally in the black box model. For the ordered searching problem, we consider queries of the type "$x_i =?$", and for the sorting and element distinctness problems, we consider queries of the type "Is $x_i < x_{i'}$?", which are simply binary comparisons. Let $H_i = \sum_{k=1}^{i} \frac{1}{k}$ denote the ith harmonic number. We prove a lower bound for each of these three problems.

Theorem 1. *Any quantum algorithm for ordered searching that errs with probability at most $\epsilon \geq 0$ requires at least*

$$\left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right)\frac{1}{\pi}(H_N - 1) \tag{1}$$

queries to the oracle. In particular, any exact quantum algorithm requires more than $\frac{1}{\pi}(\ln(N) - 1) \approx 0.220 \log_2 N$ queries.

Theorem 2. *Any comparison-based quantum algorithm for sorting that errs with probability at most $\epsilon \geq 0$ requires at least*

$$\left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right)\frac{N}{2\pi}(H_N - 1) \tag{2}$$

comparisons. In particular, any exact quantum algorithm requires more than $\frac{N}{2\pi}(\ln(N) - 1) \approx 0.110N \log_2 N$ comparisons.

Theorem 3. *Any comparison-based quantum algorithm for element distinctness that errs with probability at most $\epsilon \geq 0$ requires at least*

$$\left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right)\frac{\sqrt{N}}{2\pi}(H_N - 1) \tag{3}$$

comparisons.

The previously best known quantum lower bound for ordered searching is $\frac{1}{12}\log_2(N) - O(1)$, due to Ambainis [1]. For comparison-based sorting and element distinctness, the previously best known quantum lower bounds are respectively $\Omega(N)$ and $\Omega(\sqrt{N})$, both of which can be proven in many ways.

We prove our lower bounds by utilizing what we refer to as a weighted all-pairs inner product argument, or a probabilistic adversary argument. This proof technique is based on the work of Bennett, Bernstein, Brassard, and Vazirani [5] and Ambainis [2].

Farhi, Goldstone, Gutmann, and Sipser [12] have given an exact quantum algorithm for ordered searching using roughly $0.526 \log_2(N)$ queries. We provide an alternative quantum algorithm that is exact and uses $\log_3(N) + O(1) \approx 0.631 \log_2(N)$ queries. Our construction is radically different from the construction proposed by Farhi *et al.* [12], and these are the only constructions known leading to quantum algorithms using at most $c \log_2(N)$ queries for some constant c strictly less than 1.

Whereas most quantum algorithms are based on Fourier transforms and amplitude amplification [7], our algorithm is based on binary search trees. We initiate several applications of the binary search algorithm in quantum parallel and let them find the element we are searching for in teamwork. By cooperating, these applications can traverse the binary search tree faster than classically, hereby reducing the complexity from $\log_2(N)$ to roughly $\log_3(N)$.

There are at least three reasons why the quantum complexities of the three problems are of interest. Firstly because of their significance in algorithmics in general. Secondly because these problems possess some symmetries and periodicities of a different nature than other studied problems in quantum algorithmics. Determining symmetries and periodicities seems to be a primary ability of quantum computers and it is not at all clear how far-reaching this skill is. Thirdly because searching and sorting represent non-Boolean non-symmetric functions. A (partial) function is said to be symmetric if it is invariant under permutation of its input. Only few non-trivial quantum bounds for non-Boolean and non-symmetric functions are known.

The rest of the paper is organized as follows. We first discuss the model in Sect. 2, present our general technique for proving lower bounds in Sect. 3.1, and then apply it to the three problems in Sects. 3.2–3.4. We give our quantum algorithm for ordered searching in Sect. 4 and conclude in Sect. 5.

2 Quantum Black Box Computing

We give a formal definition of the black box model, which is slightly different from, but equivalent to, the definition of Beals, Buhrman, Cleve, Mosca, and de Wolf given in [3]. Fix some positive integer $N > 0$. The input $x = (x_0, \ldots, x_{N-1}) \in \{0,1\}^N$ is given as an oracle, and the only way we can access the bits of the oracle is via queries. A query implements the operator

$$\mathsf{O}_x : \quad |z;i\rangle \longmapsto \begin{cases} (-1)^{x_i}|z;i\rangle & \text{if } 0 \le i < N \\ |z;i\rangle & \text{if } i \ge N. \end{cases} \tag{4}$$

Here i and z are non-negative integers. By a query to oracle x we mean an application of the unitary operator O_x. We sometimes refer to O_x as the oracle.

A quantum algorithm A that uses T queries to an oracle O is a unitary operator of the form

$$\mathsf{A} = (\mathsf{UO})^T \mathsf{U}. \tag{5}$$

We always apply algorithm A on the initial state $|0\rangle$. For every integer $j \geq 0$ and every oracle x, let

$$|\psi_x^j\rangle = (\mathsf{UO}_x)^j \mathsf{U} |0\rangle \tag{6}$$

denote the state after j queries, given oracle x. After applying A, we always measure the final state in the computational basis.

Consider the computation of some function $f : S \to \{0,1\}^m$, where $S \subseteq \{0,1\}^N$. We say that algorithm A computes f with error probability bounded by ϵ, for some constant ϵ with $0 \leq \epsilon < 1/2$, if for any $x \in S$, the probability of observing $f(x)$ when the m rightmost bits of $|\psi_x^T\rangle$ are measured is at least $1 - \epsilon$.

3 Lower Bounds

3.1 General Technique

We use the notation of Sect. 2. For any $\epsilon \geq 0$, let $\epsilon' = 2\sqrt{\epsilon(1-\epsilon)}$.

The computation always starts in the same initial state $|0\rangle$, so for all oracles $x \in S$ we have $|\psi_x^0\rangle = |0\rangle$. If for two input oracles $x, y \in S$, the correct answers are different, i.e., if $f(x) \neq f(y)$, then the corresponding final states $|\psi_x^T\rangle$ and $|\psi_y^T\rangle$ must be almost orthogonal.

Lemma 4. *For all oracles $x, y \in S$ so that $f(x) \neq f(y)$, $|\langle \psi_x^T | \psi_y^T \rangle| \leq \epsilon'$.*

Now consider a probability distribution over those pairs of inputs $(x, y) \in S \times S$ for which $f(x) \neq f(y)$. For each integer $j \geq 0$, we use the following quantity to quantify the *average progress* of the algorithm in distinguishing any two inputs after applying $(\mathsf{UO})^j \mathsf{U}$,

$$W_j = \mathbf{E}_{(x,y)} \left[\langle \psi_x^j | \psi_y^j \rangle \right].$$

Observe that $W_0 = 1$ and that $W_T \leq \epsilon'$ by Lemma 4. By proving that for every j with $0 \leq j < T$, we have $|W_j - W_{j+1}| \leq \delta$, we conclude that $T \geq (1 - \epsilon')/\delta$.

For simplicity of presentation, we scale the probabilities by using a *weight function* $\omega : S \times S \to \mathbb{R}^+$. From now on, we use the following definition of W_j to quantify the overall progress of the algorithm,

$$W_j = \sum_{x,y \in S} \omega(x,y) \, \langle \psi_x^j | \psi_y^j \rangle. \tag{7}$$

Our technique is a natural generalization of Ambainis' approach [2], which uses uniform distributions over subsets of $S \times S$. Our lower bound proofs imply that non-uniform distributions can give better lower bounds. Clearly, finding a

"good" distribution is an important step in applying our technique. Another important step is to find a tight bound on the progress after each oracle query.

We end this subsection by introducing some notation and stating two lemmas we require when bounding the progress. For every $i \geq 0$, let $\mathsf{P}_i = \sum_{z \geq 0} |z;i\rangle\langle z;i|$ denote the projection operator onto the subspace querying the ith oracle bit. For $i < 0$, operator P_i is taken as the zero projection. The following lemma, which may be proven by the Cauchy–Schwarz inequality, bounds the quantified progress that one oracle query makes in distinguishing two inputs x and y.

Lemma 5. *For any oracles $x, y \in \{0,1\}^N$, and any integer $j \geq 0$,*

$$|\langle\psi_x^j|\psi_y^j\rangle - \langle\psi_x^{j+1}|\psi_y^{j+1}\rangle| \;\leq\; 2 \sum_{i:x_i \neq y_i} \|\mathsf{P}_i|\psi_x^j\rangle\| \cdot \|\mathsf{P}_i|\psi_y^j\rangle\|. \tag{8}$$

We sometimes write $|\psi_x\rangle$ as shorthand for $|\psi_x^j\rangle$ once integer j is fixed.

Let $A = [\alpha_{k,\ell}]_{1 \leq k,\ell < \infty}$ be the Hilbert matrix with $\alpha_{k,\ell} = 1/(k+\ell-1)$, and $\|\cdot\|_2$ be the spectral norm, i.e., for any complex-valued matrix $M \in \mathbb{C}^{m \times m}$, the norm $\|M\|_2$ is defined as $\max\{\|Mx\|_2\}$, where the maximum is taken over all unit vectors $x \in \mathbb{C}^m$. Let $B_N = [\beta_{k,\ell}]_{1 \leq k,\ell \leq N}$ be the matrix where entry $\beta_{k,\ell}$ is $\frac{1}{k+\ell-1}$ if $k + \ell \leq N+1$, and 0 otherwise. Clearly $\|B_N\|_2 \leq \|A\|_2$ for any $N > 0$. Our lower bound proofs rely on the following property of the Hilbert matrix.

Lemma 6 (E.g.: Choi [10]). *$\|A\|_2 = \pi$. Hence, $\|B_N\|_2 \leq \pi$.*

3.2 Lower Bound for Ordered Searching

The first non-trivial quantum lower bound on ordered searching proven was $\Omega(\sqrt{\log_2(N)}/\log_2\log_2(N))$, due to Buhrman and de Wolf [9] by an ingenious reduction from the PARITY problem. Farhi, Goldstone, Gutmann, and Sipser [11] improved this to $\log_2(N)/2\log_2\log_2(N)$, and Ambainis [1] then proved the previously best known lower bound of $\frac{1}{12}\log_2(N) - O(1)$. In [11,1], they use, as we do here, an inner product argument along the lines of [5]. In this section, we improve the lower bound by a constant factor.

For the purpose of proving the lower bound, we assume that each of the N input numbers is either 0 or 1, and that the input does not consist of all zeroes. That is, the set S of possible inputs are the ordered N-bit strings of non-zero Hamming weight. The search function $f : S \to \{0,1\}^m$ is defined by $f(x) = \min\{0 \leq i < N \mid x_i = 1\}$, where we identify the result $f(x)$ with its binary encoding as a bit-string of length $m = \lceil\log_2(N)\rceil$. As our weight function ω, we choose the inverse of the difference in Hamming weights,

$$\omega(x,y) = \begin{cases} \frac{1}{f(y)-f(x)} & \text{if } 0 \leq f(x) < f(y) < N \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

With this choice, we have that $W_0 = NH_N - N$ and by Lemma 4 also that $W_T \leq \epsilon' W_0$. Theorem 1 then follows from the next lemma.

Lemma 7. *For every j with $0 \leq j < T$ we have that $|W_j - W_{j+1}| \leq \pi N$.*

Proof. As shorthand, we write $|\psi_{f(x)}\rangle$ for $|\psi_x^j\rangle$. By Lemma 5,

$$\begin{aligned} |W_j - W_{j+1}| &\le 2 \sum_{k=0}^{N-2} \sum_{\ell=k+1}^{N-1} \frac{1}{\ell - k} \sum_{i=k}^{\ell-1} \|\mathsf{P}_i|\psi_k\rangle\| \cdot \|\mathsf{P}_i|\psi_\ell\rangle\| \\ &= 2 \sum_{d=1}^{N-1} \sum_{i=0}^{d-1} \frac{1}{d} \sum_{k=0}^{N-d-1} \|\mathsf{P}_{k+i}|\psi_k\rangle\| \cdot \|\mathsf{P}_{k+i}|\psi_{k+d}\rangle\|. \end{aligned}$$

Let vectors $\gamma = [\gamma_i]_{0 \le i < N-1} \in \mathbb{R}^{N-1}$ and $\delta = [\delta_i]_{0 \le i < N-1} \in \mathbb{R}^{N-1}$ be defined by

$$\gamma_i = \left(\sum_{k=0}^{N-1} \|\mathsf{P}_{k+i}|\psi_k\rangle\|^2 \right)^{1/2} \quad \text{and} \quad \delta_i = \left(\sum_{k=0}^{N-1} \|\mathsf{P}_{k-i-1}|\psi_k\rangle\|^2 \right)^{1/2}.$$

Then, by the Cauchy–Schwarz inequality,

$$|W_j - W_{j+1}| \le 2 \sum_{d=1}^{N-1} \sum_{i=0}^{d-1} \frac{1}{d} \gamma_i \delta_{d-i-1} = 2\gamma^t B_N \delta, \tag{10}$$

where t denotes matrix transposition. Since each vector $|\psi_k\rangle$ is of unit norm, we have $\|\gamma\|_2^2 + \|\delta\|_2^2 \le N$, so $\|\gamma\|_2 \|\delta\|_2 \le N/2$. The matrix product $2\gamma^t B_N \delta$ is upper bounded by $2\|\gamma\|_2 \cdot \|\!|B_N|\!\|_2 \cdot \|\delta\|_2$, which is at most πN by Lemma 6. □

3.3 Lower Bound for Sorting

We assume that the N numbers to be sorted, $x = (x_0, \ldots, x_{N-1})$, correspond to some permutation σ on $\{0, 1, \ldots, N-1\}$. That is, $x_i = \sigma(i)$ for every $0 \le i < N$. We assume the input to the quantum algorithm is the comparison matrix $M_\sigma = [m_{ii'}]_{0 \le i,i' < N}$ with

$$m_{ii'} = \begin{cases} 1 & \text{if } \sigma(i) < \sigma(i') \\ 0 & \text{otherwise.} \end{cases}$$

One comparison corresponds to one application of the oracle operator

$$\mathsf{O}_\sigma = \sum_{z \ge 0} \sum_{i,i' \ge 0} (-1)^{m_{ii'}} |z; i, i'\rangle\langle z; i, i'|.$$

To simplify notation, we sometimes identify the input M_σ with the underlining permutation σ.

For every pair $\{i, i'\}$ of indices with $0 \le i, i' < N$, let

$$\mathsf{P}_{ii'} = \sum_{z \ge 0} |z; i, i'\rangle\langle z; i, i'| + \sum_{z \ge 0} |z; i', i\rangle\langle z; i', i|$$

denote the projection operator onto the subspace comparing the ith and (i')th elements. For any vector $|\psi\rangle$, we use $|\psi\restriction_{\sigma,k,\ell}\rangle$ as shorthand for $\mathsf{P}_{\sigma^{-1}(k),\sigma^{-1}(\ell)}|\psi\rangle$.

For every permutation σ, and every integers $0 \leq k \leq N-2$ and $1 \leq d \leq N-1-k$, define a new permutation,

$$\sigma^{(k,d)} = (k, k+1, \ldots, k+d) \circ \sigma. \tag{11}$$

If $\tau = \sigma^{(k,d)}$, then

$$\sigma^{-1}(i) = \begin{cases} \tau^{-1}(k) & \text{if } i = k+d \\ \tau^{-1}(i+1) & \text{if } k \leq i < k+d \\ \tau^{-1}(i) & \text{otherwise.} \end{cases} \tag{12}$$

This implies that the comparison matrices M_σ and M_τ differ only on the following pairs of entries,

$$\left\{\sigma^{-1}(k+d), \sigma^{-1}(k+i)\right\} = \left\{\tau^{-1}(k), \tau^{-1}(k+i+1)\right\} \tag{13}$$

for all i with $0 \leq i < d$.

Informally, if M_σ corresponds to some list x, then M_τ corresponds to the list y obtained by replacing the element of rank $k+d$ in x by a new element of rank k (the element in x that had rank k then has rank $k+1$ in y, etc.). The only way the algorithm can distinguish σ from τ is by comparing the element of rank $k+d$ in x with one of the d elements of rank $k+i$ for some $0 \leq i < d$.

We choose the following weight function,

$$\omega(\sigma, \tau) = \begin{cases} \frac{1}{d} & \text{if } \tau = \sigma^{(k,d)} \text{ for some } k \text{ and } d \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Then one may verify that $W_0 = N!(NH_N - N)$, and $W_T \leq \epsilon' W_0$. To prove Theorem 2, we need only to prove the following lemma.

Lemma 8. *For any j with $0 \leq j < T$, $|W_j - W_{j+1}| \leq 2\pi N!$.*

Proof. Similar to the proof of Lemma 7. By Lemma 5 and (13),

$$|W_j - W_{j+1}| \leq 2 \sum_{d=1}^{N-1} \sum_{i=0}^{d-1} \frac{1}{d} \sum_{\sigma} \sum_{k=0}^{N-d-1} \left\| |\psi_\sigma\rangle_{\sigma,k+d,k+i}\rangle \right\| \cdot \left\| |\psi_{\sigma^{(k,d)}}\rangle_{\sigma,k+d,k+i}\rangle \right\|.$$

Let $\gamma = [\gamma_i]_{1 \leq i < N} \in \mathbb{R}^{N-1}$ be such that $\gamma_i = \left(\sum_\sigma \sum_{\ell=0}^{N-1} \left\| |\psi_\sigma\rangle_{\sigma,\ell,\ell+i}\rangle \right\|^2\right)^{1/2}$, where we let ℓ range from 0 to $N-1$ and simply set the thus caused undefined projection operators to be zero operators. Then by (12),

$$\sum_{\sigma} \sum_{k=0}^{N-d-1} \left\| |\psi_{\sigma^{(k,d)}}\rangle_{\sigma,k+d,k+i}\rangle \right\|^2 = \sum_{\tau} \sum_{k=0}^{N-d-1} \left\| |\psi_\tau\rangle_{\tau,k,k+i+1}\rangle \right\|^2 \leq \gamma_{i+1}^2.$$

Applying the Cauchy–Schwarz inequality, and in analogy with (10),

$$|W_j - W_{j+1}| \leq 2 \sum_{d=1}^{N-1} \sum_{i=0}^{d-1} \frac{1}{d} \gamma_{d-i} \gamma_{i+1} = 2\gamma^t B_{N-1} \gamma. \tag{15}$$

Since $\|\gamma\|_2^2 \leq N!$, we conclude that $|W_j - W_{j+1}| \leq 2\pi N!$. □

3.4 Lower Bound for Element Distinctness

We modify the adversary for sorting as follows. As in Sect. 3.3, when we talk about permutations, the underlying set is $\{0, 1, \ldots, N-1\}$.

Definition 9. *An annotated permutation is a permutation τ with a marker on a single element r_τ for some $0 \le r_\tau < N-1$.*

For every permutation σ, and every integers k and d as in Sect. 3.3, the annotated permutation $\tau = \sigma^{(k,d)}$ is the same permutation as in (11) but with the rank k element marked. The only places where M_σ and M_τ differ, are at the same entries as those in (13).

We use the same weight function as in (14). Then $W_0 = N!(NH_N - N)$ and $W_T \le \epsilon' W_0$. We need only to prove the following lemma.

Lemma 10. *For any integer j with $0 \le j < T$, $|W_j - W_{j+1}| \le 2\pi N!\sqrt{N}$.*

Proof. Almost identical to the proof for Lemma 8, except that we now require a second vector $\delta = [\delta_i]_{1 \le i < N} \in \mathbb{R}^{N-1}$ with $\delta_i = \left(\sum_\tau \left\| |\psi_\tau\rangle_{\tau, r_\tau, r_\tau + i} \right\|^2\right)^{1/2}$. Then by (12),

$$\sum_\sigma \sum_{k=0}^{N-d-1} \left\| |\psi_{\sigma^{(k,d)}}\rangle_{\sigma, k+d, k+i} \right\|^2 = \sum_{\tau : r_\tau < N-d} \left\| |\psi_\tau\rangle_{\tau, r_\tau, r_\tau + i + 1} \right\|^2 \le \delta_{i+1}^2.$$

In analogy with (15), we have

$$|W_j - W_{j+1}| \le 2 \sum_{d=1}^{N-1} \sum_{i=0}^{d-1} \frac{1}{d} \gamma_{d-i} \delta_{i+1} = 2\gamma^t B_{N-1} \delta.$$

Besides having $\|\gamma\|^2 \le N!$ as in the proof of Lemma 8, we also have that

$$\|\delta\|^2 = \sum_{i=1}^{N-1} \sum_\tau \left\| |\psi_\tau\rangle_{\tau, r_\tau, r_\tau + i} \right\|^2 \le N!(N-1) \le N!N.$$

Therefore, $|W_j - W_{j+1}| \le 2\pi\sqrt{N!}\sqrt{N!N} = 2\pi N!\sqrt{N}$. □

4 A $\log_3(N)$ Algorithm for Ordered Searching

We begin by considering binary search trees on which our quantum algorithm is based. Let $\mathcal{T}$ be a binary tree with $N \ge 2$ leaves. We put colored pebbles on the (internal) vertices of $\mathcal{T}$ subject to the following 2 conditions:

(A) on every path from the root of $\mathcal{T}$ to a leaf, there is exactly 1 pebble of each color, and
(B) the number of pebbles p_v on any vertex $v \in \mathcal{T}$ is at least as large as the total number of pebbles on its proper ancestors.

We say that $\mathcal{T}$ is *covered by N' pebbles* if we can satisfy the 2 above conditions using at most N' pebbles of each color. We want to minimize the maximum number N' of pebbles used of any color. We say a covering is *fair* if it uses the same number of pebbles of every color. We say a covering is *tight* if, for all vertices $v \in \mathcal{T}$, we have that p_v equals the total number of pebbles on its proper ancestors, or there are no pebbles on any of the ancestors of v. We require the following two lemmas.

Lemma 11. *For every even integer $N \geq 2$, there exists a binary tree with N leaves that can be fairly and tightly covered by $N' = \lfloor \frac{1}{3}N + \log_2(N) \rfloor$ pebbles using 2^s colors, where $s = \lfloor \log_4(N/2) \rfloor$.*

Lemma 12. *Let integer-valued function $\tilde{F}$ be recursively defined by*

$$\tilde{F}(N) = \begin{cases} \tilde{F}\big(\lfloor \frac{1}{3}N + \log_2(N) + 1 \rfloor\big) + 1 & \text{if } N > 8 \\ 1 & \text{if } N \leq 8. \end{cases}$$

Then $\tilde{F}(N) = \log_3(N) + O(1)$.

As in Sect. 3.2, we assume the oracle $x = (x_0, \ldots, x_{N-1}) \in \{0,1\}^N$ is a binary string of non-zero Hamming weight. The problem is to determine the leftmost 1 in x, that is, to compute $f(x) = \min\{0 \leq i < N \mid x_i = 1\}$. Let $\mathcal{T}$ be a binary tree with N leaves for which Lemma 11 holds. Let $s = \lfloor \log_4(N/2) \rfloor$ and $N' = \lfloor \frac{1}{3}N + \log_2(N) \rfloor$ be as in the lemma. We label the N leaves of $\mathcal{T}$ by $\{0, \ldots, N-1\}$ from left to right. Let $\ell_{f(x)}$ denote the leaf labelled by $f(x)$, and let $\mathcal{P}$ denote the path from the root of $\mathcal{T}$ to the parent of $\ell_{f(x)}$. We think of $\mathcal{P}$ as the path the classical search algorithm would traverse if searching for $f(x)$ in tree $\mathcal{T}$.

Let $\mathcal{C} = \{c_0, \ldots, c_{2^s-1}\}$ be the set of 2^s colors used in Lemma 11. For each color $c \in \mathcal{C}$, let V_c denote the set of vertices in $\mathcal{T}$ populated by a pebble of color c. By Condition (A), there are at most N' such vertices, that is, $|V_c| \leq N'$. Let v_c denote the unique vertex in V_c that is on path $\mathcal{P}$. We think of vertex v_c as the root of the subtree "containing" leaf $\ell_{f(x)}$. Note that, by definition, $v_c \in \mathcal{P}$ for every color $c \in \mathcal{C}$, and that $\sum_{v \in \mathcal{P}} p_v = 2^s$ by Condition (A).

Our algorithm utilizes 3 unitary operators, U_1, O'_x, and U_2. The first operator, U_1, is defined by

$$\mathsf{U}_1 : \quad |v\rangle|0\rangle \longmapsto |v\rangle\left(\frac{1}{\sqrt{p_v}} \sum_c |c\rangle\right) \qquad (v \in \mathcal{T}), \tag{16}$$

where the summation is over all colors $c \in \mathcal{C}$ that are represented by a pebble on vertex v. We refer to U_1 as the *coloring operator* and its inverse as the *un-coloring operator*.

The query operator O'_x is defined by

$$\mathsf{O}'_x : \ |v\rangle \longmapsto \begin{cases} |v; x_i\rangle & \text{if there are no pebbles on the parent of } v \\ (-1)^{x_i}|v\rangle & \text{otherwise,} \end{cases} \tag{17}$$

where i denotes the label of the rightmost leaf in the left subtree of vertex v. Query operator O'_x is clearly unitary (or rather, can be extended to a unitary operator since it is only defined on a proper subspace). Operator O'_x is slightly different from, but equivalent to, the query operator defined in Sect. 2. It mimics the classical search algorithm by querying the bit x_i that corresponds to the rightmost leaf in the left subtree of v.

We also use a unitary operator U_2 that maps each vertex to a superposition over the leaves in its subtree. For every vertex and leaf u in $\mathcal{T}$, let $\mathcal{L}(u)$ denote the set of leaves in the subtree rooted at u, and let

$$|\Phi_u\rangle = \sum_{\ell \in \mathcal{L}(u)} \frac{1}{\sqrt{2^{d(u,\ell)}}} |\ell\rangle, \tag{18}$$

where $d(u, \ell)$ denotes the absolute value of the difference in depths of u and leaf ℓ. The unitary operator U_2 is (partially) defined as follows. For every vertex $v \in \mathcal{T}$ with no pebbles on its parent,

$$|v\,;0\rangle \;\longmapsto\; |\Phi_{\mathrm{right}(v)}\rangle \tag{19.1}$$

$$|v\,;1\rangle \;\longmapsto\; |\Phi_{\mathrm{left}(v)}\rangle, \tag{19.2}$$

and for every vertex $v \in \mathcal{T}$ with pebbles on its parent,

$$|v\rangle \;\longmapsto\; \frac{1}{\sqrt{2}}\big(|\Phi_{\mathrm{right}(v)}\rangle - |\Phi_{\mathrm{left}(v)}\rangle\big). \tag{19.3}$$

Here $\mathrm{left}(v)$ denotes the left child of v, and $\mathrm{right}(v)$ the right child.

Our quantum algorithm starts in the initial state $|0\rangle$ and produces the final state $|\ell_{f(x)}\rangle$. Let $F(N)$ denote the number of queries used by the algorithm on an oracle x of size N.

1. We first set up a superposition over all 2^s colors, $\frac{1}{\sqrt{2^s}} \sum_{c \in \mathcal{C}} |0\rangle|c\rangle$.
2. We then apply our exact quantum search algorithm recursively. For each color $c \in \mathcal{C}$ in quantum parallel, we search recursively among the vertices in V_c, hereby determining the root $v_c \in V_c$ of the subtree containing the leaf $\ell_{f(x)}$. Since $|V_c| \le N'$, this requires at most $F(N'+1)$ queries to oracle x and produces the superposition $\frac{1}{\sqrt{2^s}} \sum_{c \in \mathcal{C}} |v_c\rangle|c\rangle$. Since every vertex v_c in this sum is on the path $\mathcal{P}$, we can rewrite the sum as
$$\frac{1}{\sqrt{2^s}} \sum_{v \in \mathcal{P}} |v\rangle \sum_{c \in \mathcal{C}: v_c = v} |c\rangle.$$
3. We then apply the un-coloring operator U_1^{-1}, producing the superposition $\frac{1}{\sqrt{2^s}} \sum_{v \in \mathcal{P}} \sqrt{p_v}\,|v\rangle|0\rangle$. Ignoring the second register which always holds a zero, this is
$$\frac{1}{\sqrt{2^s}} \sum_{v \in \mathcal{P}} \sqrt{p_v}\,|v\rangle.$$
That is, we have (recursively) obtained a superposition over the vertices on the path $\mathcal{P}$ from the root of $\mathcal{T}$ to the parent of the leaf $\ell_{f(x)}$ labelled by $f(x)$.

4. We then apply the operator $\mathsf{U}_2\mathsf{O}'_x$, producing the final state

$$\mathsf{U}_2\mathsf{O}'_x \frac{1}{\sqrt{2^s}} \sum_{v\in\mathcal{P}} \sqrt{p_v}\,|v\rangle \;=\; \frac{1}{\sqrt{2^s}} \sum_{v\in\mathcal{P}} \sqrt{p_v}\,\mathsf{U}_2\mathsf{O}'_x\,|v\rangle,$$

which one can show equal to $|\ell_{f(x)}\rangle$. Thus, a final measurement of this state yields $f(x)$ with certainty.

The total number of queries to the oracle x is at most $F(N'+1)+1$, and thus, by Lemma 12, the algorithm uses at most $\log_3(N) + O(1)$ queries. Theorem 13 follows.

Theorem 13. *The above described quantum algorithm for searching an ordered list of N elements is exact and uses at most $\log_3(N) + O(1)$ queries.*

5 Concluding Remarks and Open Problems

The inner product of two quantum states is a measure for their distinguishability. We have proposed a weighted all-pairs inner product argument as a tool for proving lower bounds in the quantum black box model. The possibility of using non-uniform weights seems particularly suitable when proving lower bounds for non-symmetric (possibly partial) functions. It could be interesting to consider other measures than inner products, as discussed, for instance, by Zalka [18], Jozsa and Schlienz [15], and Vedral [17].

The result of Grigoriev, Karpinski, Meyer auf der Heide, and Smolensky [13] implies that if only comparisons are allowed, the randomized decision tree complexity of element distinctness has the same $\Omega(N \log N)$ lower bound as sorting. Interestingly, their quantum complexities differ dramatically: the quantum algorithm by Buhrman *et al.* [8] uses only $O(N^{3/4} \log N)$ comparisons. There is still a big gap between this upper bound and our lower bound of $\Omega(N^{1/2} \log N)$. One way of closing this gab might be to consider quantum time-space tradeoffs, as has been done for the classical case [6,4].

Our algorithm for searching an ordered list with complexity $\log_3(N)+O(1)$ is based on the classical binary search algorithm. The quantum algorithm initiates several independent walks/searches at the root of the binary search tree. These searches traverse down the tree faster than classically by cooperating, and they eventually all reach the leaf we are searching for in roughly $\log_3(N)$ steps. It could be interesting to consider if similar ideas can be used to speed up other classical algorithms. For instance one may consider other applications of operators like U_2 acting on rooted trees and graphs.

Acknowledgements. We are grateful to Andris Ambainis, Harry Buhrman, Mark Ettinger, Gudmund S. Frandsen, Dieter van Melkebeek, Hein Röhrig, Daniel Wang, Ronald de Wolf, Andy Yao, and especially Sanjeev Arora, for their precious comments and suggestions.

References

1. Ambainis, A.: A better lower bound for quantum algorithms searching an ordered list. Proc. of 40th IEEE FOCS (1999) 352–357
2. Ambainis, A.: Quantum lower bounds by quantum arguments. Proc. of 32nd ACM STOC (2000) 636–643
3. Beals, R., Buhrman, H., Cleve, R., Mosca, M., de Wolf, R.: Quantum lower bounds by polynomials. Proc. of 39th IEEE FOCS (1998) 352–361
4. Beame, P.: A general sequential time-space tradeoff for finding unique elements. SIAM J. Comput. **20** (1991) 270–277
5. Bennett, C. H., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computation. SIAM J. Comput. **26** (1997) 1510–1523
6. Borodin, A., Fischer, M. J., Kirkpatrick, D. G., Lynch, N A., Tompa, M.: A time-space tradeoff for sorting on nonoblivious machines. J. Comput. Sys. Sci. **22** (1981) 351–364
7. Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. quant-ph/0005055, 2000
8. Buhrman, H., Dürr, C., Heiligman, M., Høyer, P., Magniez, F., Santha, M., de Wolf, R.: Quantum algorithms for element distinctness. Proc. of 16th IEEE Computational Complexity (2001) (to appear)
9. Buhrman, H., de Wolf, R.: A lower bound for quantum search of an ordered list. Inform. Proc. Lett. **70** (1999) 205–209
10. Choi, M.-D.: Tricks or treats with the Hilbert matrix. Amer. Math. Monthly **90** (1983) 301–312
11. Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: A limit on the speed of quantum computation for insertion into an ordered list. quant-ph/9812057, 1998
12. Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: Invariant quantum algorithms for insertion into an ordered list. quant-ph/9901059, 1999
13. Grigoriev, D., Karpinski, M., Meyer auf der Heide, F., Smolensky, R.: A lower bound for randomized algebraic decision trees. Comput. Complexity **6** (1996/1997) 357–375
14. Grover, L. K.: Quantum mechanics helps in searching for a needle in a haystack. Phys. Rev. Letters **79** (1997) 325–328
15. Jozsa, R., Schlienz, J.: Distinguishability of states and von Neumann entropy. Phys. Rev. A **62** (2000) 012301
16. Shor, P. W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26** (1997) 1484–1509
17. Vedral, V.: The role of relative entropy in quantum information theory. quant-ph/0102094, 2001
18. Zalka, Ch.: Grover's quantum searching algorithm is optimal. Phys. Rev. A **60** (1999) 2746–2751

Many of the above references can be found at the Los Alamos National Laboratory e-print archive (http://arXiv.org/archive/quant-ph).

Lower Bounds in the Quantum Cell Probe Model

Pranab Sen[1] and S. Venkatesh[2]

[1] School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai 400005, India.
pranab@tcs.tifr.res.in***.
[2] School of Mathematics, Institute for Advanced Study, Princeton, NJ 08540, USA.
venkat@ias.edu†.

Abstract. We introduce a new model for studying quantum data structure problems — the *quantum cell probe model*. We prove a lower bound for the *static predecessor* problem in the *address-only* version of this model where, essentially, we allow quantum parallelism only over the 'address lines' of the queries. This model subsumes the classical cell probe model, and many quantum query algorithms like Grover's algorithm fall into this framework. We prove our lower bound by obtaining a round elimination lemma for quantum communication complexity. A similar lemma was proved by Miltersen, Nisan, Safra and Wigderson [9] for classical communication complexity, but their proof does not generalise to the quantum setting.
We also study the *static membership* problem in the quantum cell probe model. Generalising a result of Yao [16], we show that if the storage scheme is *implicit*, that is it can only store members of the subset and 'pointers', then any quantum query scheme must make $\Omega(\log n)$ probes. We also consider the one-round quantum communication complexity of set membership and show tight bounds.

1 Introduction

A static data structure problem consists of a set of data D, a set of queries Q, a set of answers A, and a function $f : D \times Q \to A$. The aim is to store the data efficiently and succinctly, so that any query can be answered with only a few probes to the data structure. In a seminal paper [16], Yao introduced the (classical) *cell probe model* for studying static data structure problems in the classical setting. Thereafter, this model has been used extensively to prove upper and lower bounds for several data structure problems (see [4], [1], [9], [2]). A classical (s, w, t) *cell probe scheme* for f has two components: a *storage scheme* and a *query scheme*. Given the data to be stored, the storage scheme stores it as a table of s cells, each cell w bits long. The query scheme has to answer

*** Part of this work was done while visiting UC Berkeley and DIMACS, under a Sarojini Damodaran International Fellowship grant.
† Supported by NSF grant CCR–9987845 and a joint IAS-DIMACS postdoctoral fellowship.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 358–369, 2001.

queries about the data stored. Given a query, the query scheme computes the answer to that query by making at most t probes to the stored table, where each probe reads one cell at a time. The storage scheme is deterministic whereas the query scheme can be deterministic or randomised. The goal is to study tradeoffs between s, t and w. For an overview of results in this model, see the survey by Miltersen [8].

In this paper, we study static data structure problems, such as the *static membership problem* and the *static predecessor problem*, when the algorithm is allowed to query the table using a quantum superposition. We formalise this by defining the *quantum cell probe model* similar to the *quantum bit probe model* of [13]. We show a lower bound for the predecessor problem in a restricted version of this model, which we call the *address-only* quantum cell probe model. In the predecessor problem, the storage scheme has to store a subset S of size at most n from the universe $[m]$, such that given any query element $x \in [m]$, one can quickly find the predecessor of x in S.

Result 1 (Lower bound for predecessor): Suppose that we have an address-only quantum cell probe solution to the static predecessor problem, where the universe size is m and the subset size is at most n, using $n^{O(1)}$ cells each containing $O(\log m)$ bits. Then the number of queries is at least $\Omega(\sqrt{\log \log m})$ as a function of m, and at least $\Omega(\log^{1/3} n)$ as a function of n.

We then consider the static membership problem. Here one has to answer membership queries instead of predecessor queries. Yao [16] showed that if the universe is large enough, any classical deterministic *implicit* scheme for the static membership problem must make $\Omega(\log n)$ probes to the table in the worst case. An implicit scheme either stores a 'pointer value' (viz. a value which is not an element of the universe) or an element of S in a cell. In particular, it is not allowed to store an element of the universe which is not a member of S. We generalise Yao's result to the quantum setting.

Result 2 (informal statement): *If the storage scheme is implicit then, if the universe is large enough compared to the number of cells of storage, the quantum query algorithm must make $\Omega(\log n)$ probes.*

Remarks:

1. Our *address-only* quantum cell probe model subsumes the classical cell probe model. Hence, our lower bound for the static predecessor problem is a generalisation of a similar result shown for the classical cell probe model with randomised query schemes, by Miltersen et al [9]. This lower bound is the best known for classical randomised query schemes, if the storage scheme uses $n^{O(1)}$ cells each containing $O(\log m)$ bits. Thus, our quantum lower bounds are as strong as the best known classical randomised lower bounds. The best upper bound known uses $O(n)$ cells of storage, each cell contains $O(\log m)$ bits, and answers predecessor queries with $O(\min(\log \log m / \log \log \log m, \sqrt{\log n / \log \log n}))$ probes. In

fact, it is a classical deterministic query scheme. For deterministic schemes, the above bound is tight. Both the above bound, and its optimality for deterministic schemes, have been proved by Beame and Fich [1].

2. It is known that querying in superposition gives a speed up over classical algorithms for certain data retrieval problems, the most notable one being Grover's algorithm [5] for searching an unordered list of n elements using $O(\sqrt{n})$ quantum queries. The power of quantum querying for data structure problems was studied in the context of static membership by Radhakrishnan et al. [13]. In their paper, they worked in the quantum bit probe model, which is our quantum cell probe model where the cell size is just one bit. They showed, roughly speaking, that quantum querying does not give much advantage over classical schemes for the set membership problem. Our result above seems to suggest that quantum search is perhaps not more powerful than classical search for the predecessor problem as well.

3. In the next section, we formally describe the "address-only" restrictions we impose on the query algorithm. Informally, they amount to this: *we allow quantum parallelism over the 'address lines' going into the table, but we have a fixed quantum state on the 'data lines'.* This restriction on quantum querying does not make the problem trivial. In fact, many non-trivial quantum search algorithms, such as Grover's algorithm [5] and Hoyer and Neerbek's algorithm [6], already satisfy these restrictions.

4. For the static membership problem, Fredman, Komlós and Szemerédi [4] have shown a classical deterministic cell probe solution where the storage scheme uses $O(n)$ cells each containing $O(\log m)$ bits, and the query scheme makes only a constant number of probes. In this solution, the storage scheme may store elements of the universe in the table which are not members of the subset to be stored. Hence the restriction that the storage scheme be implicit is necessary for any such result. We note that implicit storage schemes include many of the standard storage schemes like sorted array, hash table, search trees etc.

1.1 Techniques

The lower bounds for the static membership problem shown in the quantum bit probe model [13], relied on linear algebraic techniques. Unfortunately, these techniques appear to be powerless for the quantum cell probe model. In fact, to show the lower bound above for the static predecessor problem, we use a connection between quantum data structure problems and two-party quantum communication complexity, similar to what was used by Miltersen, Nisan, Safra and Wigderson [9] for showing the classical lower bound. They proved a technical lemma in classical communication complexity called the *round elimination lemma* and derived from it lower bounds for various static data structure problems. In this paper we prove an analogue of their round elimination lemma for the quantum communication complexity model, which we then use to show the quantum lower bound for the static predecessor problem. The quantum round elimination lemma also has applications to other quantum communication complexity problems, which might be interesting on their own.

Suppose $f : X \times Y \to Z$ is a function. In the communication game corresponding to f, Alice gets a string $x \in X$, Bob gets a string $y \in Y$ and they have to compute $f(x, y)$. In the communication game corresponding to $f^{(n)}$, Alice gets n strings $x_1, \ldots, x_n \in X$; Bob gets an integer $i \in [n]$, a string $y \in Y$, and a copy of the strings $x_1, \ldots, x_{i-1}$. Their aim is to compute $f(x_i, y)$. Suppose a protocol for $f^{(n)}$ is given where Alice starts, and her first message is a bits long, where a is much smaller than n. Intuitively, it would seem that since Alice does not know i, the first round of communication cannot give much information about x_i, and thus, would not be very useful to Bob. The round elimination lemma justifies this intuition. Moreover, we show that this is true even if Bob also gets copies of $x_1, \ldots, x_{i-1}$, a case which is needed in many data structure applications.

Result 3 (Quantum round elimination lemma, informal statement): *A t round quantum protocol for $f^{(n)}$ with Alice starting, gives us a $t-1$ round protocol for f with Bob starting, with similar message complexity and error probability.*

Round reduction arguments have been given earlier in quantum communication complexity, most notably by Nayak, Ta-Shma and Zuckerman [10]. However, for technical reasons, the previous arguments do not go far enough to prove lower bounds for the communication games arising from data structure problems like the predecessor problem. We need a technical quantum version of the round elimination lemma of Miltersen et al [9], to prove the desired lower bounds.

We also study the set membership communication game $\mathrm{MEM}_{m,n}$, where Alice is given an element x of a universe of size m, and Bob is given a subset S of the universe of size at most n. They have to communicate and decide whether $x \in S$. We consider bounded error one round quantum communication protocols for this problem in both the cases of Alice and Bob speaking. We give tight upper and lower bounds for this problem in both these cases.

Result 4: *The bounded error one round quantum communication complexity of the set membership problem $MEM_{m,n}$, when Alice starts, is $\Theta(\log n + \log\log m)$, and when Bob starts, is $\Theta(n + \log\log m)$.*

1.2 Organisation of the Paper

Section 2 contains definitions of various terms that will be used throughout the paper. In Section 3, we discuss some lemmas that will be needed in the proofs of the main theorems. Section 4 contains the proof of the quantum round elimination lemma. In Section 5, we prove our quantum lower bounds for the static predecessor problem. The proofs of our results on implicit storage schemes for the static membership problem, and the one round quantum communication complexity of set membership, as well as proofs of various lemmas which have been omitted due to lack of space, can be found in the full version [14].

2 Definitions

In this section we define some of the terms which we will be using in this paper.

2.1 The Quantum Cell Probe Model

A quantum (s, w, t) *cell probe scheme* for a static data structure problem $f : D \times Q \to A$ has two components: a classical deterministic *storage scheme* that stores the data $d \in D$ in a table T_d using s cells each containing w bits, and a quantum *query scheme* that answers queries by 'quantumly probing a cell at a time' at most t times. Formally speaking, the table T_d for the stored data is made available to the query algorithm in the form of an oracle unitary transform O_d. To define O_d formally, we represent the basis states of the query algorithm as $|j, b, z\rangle$, where $j \in \{0, \ldots, s-1\}$ is a binary string of length $\log s$, b is a binary string of length w, and z is a binary string of some fixed length. Here, j denotes the address of a cell in the table T_d, b denotes the qubits which will hold the contents of a cell and z stands for the rest of the qubits in the query algorithm. O_d maps $|j, b, z\rangle$ to $|j, b \oplus (T_d)_j, z\rangle$, where $(T_d)_j$ is a bit string of length w and denotes the contents of the jth cell in T_d. A quantum query scheme with t probes is just a sequence of unitary transformations

$$U_0 \to O_d \to U_1 \to O_d \to \ldots U_{t-1} \to O_d \to U_t$$

where U_j's are arbitrary unitary transformations that do not depend on the data stored. For a query $q \in Q$, the computation starts in an observational basis state $|q\rangle|0\rangle$, where we assume that the ancilla qubits are initially in the basis state $|0\rangle$. Then we apply in succession, the operators $U_0, O_d, \ldots, O_d, U_t$, and measure the final state. The answer consists of the values on some of the output wires of the circuit. We require that the answer be correct with probability at least $2/3$.

We now formally define the *address-only quantum cell probe model*. Here the storage scheme is as in the general model, but the query scheme is restricted to be 'address-only'. This means that the state vector before a query to the oracle is always a *tensor product* of a state vector on the address and work qubits (the $|j, z\rangle$ part in $|j, b, z\rangle$ above), which can depend on the query element and the probe number, and a state vector on the data qubits (the $|b\rangle$ part in $|j, b, z\rangle$ above), which is *independent of the query element* but can vary with the probe number. Intuitively, we are only making use of quantum parallelism over the address lines. This mode of querying a table subsumes classical querying, and also many non-trivial quantum algorithms like Grover's algorithm [5], Hoyer and Neerbek's algorithm [6] etc. satisfy this condition. For Grover, and Hoyer and Neerbek, the state vector on the data qubit is $(|0\rangle - |1\rangle)/\sqrt{2}$, independent of the probe number.

2.2 Quantum Communication Protocols

We consider two party quantum communication protocols as defined by Yao [17]. Suppose $f : X \times Y \to Z$ is a function. In the communication game corresponding to f, Alice gets a string $x \in X$, Bob gets a string $y \in Y$ and they have to compute $f(x, y)$. We say a quantum protocol computes f with ϵ-error, if for any input $(x, y) \in X \times Y$, the probability that the protocol outputs the correct result

$f(x, y)$ is at least $1 - \epsilon$. The term 'bounded error quantum protocol' means that $\epsilon = 1/3$.

We require that Alice and Bob make a secure copy of their inputs before beginning the protocol. This is possible since the inputs to Alice and Bob are in computational basis states. Thus the qubits of Alice and Bob holding their inputs are never sent as messages, remain unchanged throughout the protocol and are never measured i.e. some work qubits are measured to determine the result of the protocol. We call such protocols *secure*. We will assume henceforth that all our protocols are secure.

We now define a class of quantum protocols called *safe* protocols, which will be used in the statement of the round elimination lemma.

Definition 1 (Safe quantum protocol). *A $[t, c, a, b]^A$ safe quantum protocol P is a secure protocol where the per round message lengths of Alice and Bob are a and b qubits respectively, Alice starts first and the communication goes on for t rounds. The notation $[t, c, a, b]^B$ means the same as above, except that Bob starts first. We allow the first message to have an overhead of c qubits i.e. if Alice starts, the first message is $a + c$ qubits long and if Bob starts, the first message is $b + c$ qubits long. The density matrix of the overhead is independent of the inputs to Alice and Bob. If $c = 0$, we abbreviate the notation to a $[t, a, b]^A$ protocol.*

Remark: The concept of a safe quantum protocol may look strange at first. The reason we need to define it, intuitively speaking, is as follows. The communication games arising from data structure problems often have an asymmetry between the message lengths of Alice and Bob. This asymmetry is crucial to prove lower bounds on the number of rounds of communication. In the previous quantum round reduction arguments, the complexity of the first message in the protocol increases quickly as the number of rounds is reduced and the asymmetry gets lost. This leads to a problem where the first message soon gets big enough to potentially convey substantial information about the input of one player to the other, destroying any hope of proving strong lower bounds on the number of rounds. The concept of a safe protocol allows us to get around this problem. We show through a careful quantum information theoretic analysis of the round reduction process, that in a safe protocol, though the complexity of the first message increases a lot, this increase is confined to the safe overhead and so, the information content does not increase much. This gives us an asymmetry in the *information flow*. This is sufficient to let the round elimination arguments go through in various applications.

In this paper we will deal with quantum protocols with *public coins*. Intuitively, a public coin quantum protocol is a probability distribution over (*coinless*) quantum protocols. We shall henceforth call the standard definition of a quantum protocol as *coinless*. Our definition is similar to the classical scenario, where a randomised protocol with public coins is a probability distribution over deterministic protocols. We note however, that our definition of a public coin quantum protocol is *not* the same as that of a quantum protocol with prior en-

tanglement, which has been studied previously (see e.g. [3]). Our definition is weaker, in that it does not allow the unitary transformations of Alice and Bob to alter the 'public coin'.

Definition 2 (Public coin quantum protocol). *In a quantum protocol with a public coin, there is a shared quantum state called a* public coin*, of the form* $\sum_c \sqrt{p_c}|c\rangle$*, where* p_c *are non-negative real numbers and* $\sum p_c = 1$*. Alice and Bob make a secure copy of the coin before commencing the protocol. Thus, if the coin is in a basis state* $|c\rangle$*, the unitary transformations of Alice and Bob do not alter it. The coin is never measured.*

Hence, one can think of the public coin quantum protocol to be a probability distribution, with probability p_c, over coinless quantum protocols indexed by the coin basis states c. A *safe public coin* quantum protocol is thus, a probability distribution over safe coinless quantum protocols.

Remark: We need to define public coin quantum protocols, so as to make use of the harder direction of Yao's minimax lemma [15]. The minimax lemma is the main tool which allows us to convert average case round reduction arguments to worst case arguments. We need worst case type round reduction arguments in proving lower bounds for the rounds complexity of communication games arising from data structure applications. This is because many of these lower bound proofs use some notion of "self-reducibility", arising from the original data structure problem, which fails to hold in the average case.

For an input x, y, we define the error $\epsilon^P_{x,y}$ of the (coinless or public coin) protocol P to be the probability that the result of P on x, y is not equal to $f(x, y)$. For a coinless quantum protocol P, given a probability distribution μ on the inputs x, y of a specified size, we define the average error ϵ^P_μ of P with respect to μ as the expectation over μ of the error of P on inputs x, y. We define ϵ^P to be worst case error of P on inputs x, y.

In the proof of the round elimination lemma, we need to do parallel repetitions of public coin protocols. We also construct new protocols from old ones using both the directions of Yao's minimax lemma. We note that all these operations preserve the "safety" of the protocol.

3 Preliminaries

In this section we state some facts which will be useful in what follows.

3.1 Quantum Cell Probe Complexity and Communication

In this subsection, we describe the connection between the quantum cell probe complexity of a static data structure problem and the quantum communication complexity of an associated communication game. Let $f : D \times Q \to A$ be a static data structure problem. Consider a two-party communication problem where Alice is given a query $q \in Q$, Bob is given data $d \in D$, and they have to communicate and find out the answer $f(d, q)$. We have the following lemma.

Lemma 1. *Suppose we have a quantum (s, w, t)-cell probe solution to the static data structure problem f. Then we have a $[2t, \log s + w, \log s + w]^A$ safe coinless quantum protocol for the corresponding communication problem. If the query scheme is address-only, we can get a $[2t, \log s, \log s + w]^A$ safe coinless quantum protocol.*

Proof. The protocol just simulates the cell probe solution. Note that if the query scheme is address-only, the messages from Alice to Bob need consist only of the 'address' part. The details are omitted. □

3.2 Background from Quantum Information Theory

In this subsection, we discuss some basic facts from quantum information theory that will be used in the proof of the round elimination lemma. We follow the notation of Nayak, Ta-Shma and Zuckerman's paper [10]. For a good account of quantum information theory, see the book by Nielsen and Chuang [12].

If A is a quantum system with density matrix ρ, then $S(A) \stackrel{\Delta}{=} S(\rho) \stackrel{\Delta}{=} -\mathrm{Tr}\, \rho \log \rho$ is the *von Neumann entropy* of A. If A, B are two disjoint quantum systems, their *mutual information* is defined as $I(A : B) \stackrel{\Delta}{=} S(A)+S(B)-S(AB)$.

Suppose X is a classical random variable. Let X be in a mixed state $\{p_x, |x\rangle\}$, $|x\rangle$ orthonormal. Let Q be a quantum encoding of X i.e. it is an encoding $|x\rangle \mapsto \sigma_x$, where σ_x is a density matrix. Thus, the joint density matrix of (X, Q) is $\sum_x p_x |x\rangle\langle x| \otimes \sigma_x$. Define $\sigma \stackrel{\Delta}{=} \sum_x p_x \sigma_x$ to be the density matrix of the average encoding. Then, $S(XQ) = S(X) + \sum_x p_x S(\sigma_x)$, and hence, $I(X : Q) = S(\sigma) - \sum_x p_x S(\sigma_x)$.

If X can be written as $X = X_1 X_2$, where X_1, X_2 are classical random variables, and Q is a quantum encoding of X, we can define $I((X_1 : Q)|X_2 = x_2)$ to be the mutual information between X_1 and Q when X_2 is fixed to x_2.

We now state the following propositions, whose proofs are to be found in the full version [14].

Proposition 1. *Suppose $M = M_1 M_2$ is a quantum encoding of a classical random variable X, where the density matrix of M_2 is independent of X. Let M_1 be supported on a qubits. Then, $I(X : M) \leq 2a$.*

Proposition 2. *Suppose M is a quantum encoding of a classical random variable X. Suppose $X = X_1 X_2 \ldots X_n$, where the X_i are classical independent random variables. Then, $I(X_1 \ldots X_n : M) = \sum_{i=1}^{n} I(X_i : M X_1 \ldots X_{i-1})$.*

Proposition 3. *Let X, Y be classical random variables and M be a quantum encoding of (X, Y). Then $I(Y : MX) = I(X : Y) + E_X[I((Y : M)|X = x)]$.*

We use the *trace norm* on linear operators to measure the "distance" between two density matrices. For a linear operator A, the trace norm of A is defined as $\|A\|_t \stackrel{\Delta}{=} \mathrm{Tr}\, \sqrt{A^\dagger A}$. The trace distance between two density matrices ρ_1, ρ_2,

$\|\rho_1 - \rho_2\|_t$, bounds the ℓ_1 distance between the probability distributions on the outcomes obtained by a measurement on ρ_1 and ρ_2.

In the proof of the round elimination lemma, we will use the "average encoding theorem" in the strong form by Klauck [7]. We state it below in the version required for our purposes, for completeness. A short proof sketch of the theorem can be found in the full version [14].

Theorem 1 (Average encoding theorem). *Let X, Q be two disjoint quantum systems where X is a classical random variable, which takes value x with probability p_x, and Q is a quantum encoding $x \mapsto \sigma_x$ of X. Let the density matrix of the average encoding be $\sigma \stackrel{\Delta}{=} \sum_x p_x \sigma_x$. Then*

$$\sum_x p_x \|\sigma_x - \sigma\|_t \leq \sqrt{(2 \ln 2) I(X : Q)}$$

4 The Quantum Round Elimination Lemma

In this section we prove our round elimination lemma for safe public coin quantum protocols. Since a public coin quantum protocol can be converted to a coinless protocol at the expense of an additional "safe" overhead in the first message, we also get a similar round elimination lemma for coinless protocols. We can decrease the overhead to logarithmic in the total bit size of the inputs by a technique similar to the public to private coins conversion for classical randomised protocols [11]. But since the statement of the round elimination lemma is cleanest for safe public coin quantum protocols, we give it below in this form only.

Lemma 2 (Round elimination lemma). *Suppose $f : X \times Y \to Z$ is a function. Suppose the communication game $f^{(n)}$ has a $[t, c, a, b]^A$ safe public coin quantum protocol with worst case error at most δ. Then there is a $[t-1, c+a, a, b]^B$ safe public coin quantum protocol for f with worst case error at most $\epsilon \stackrel{\Delta}{=} 2\delta + 2(8a \ln 2/n)^{1/4}$.*

Proof. By the harder direction of Yao's minimax lemma [15], it suffices to give, for any probability distribution D on $X \times Y$, a $[t-1, c+a, a, b]^B$ safe coinless quantum protocol P for f with average distributional error $\epsilon_D^P \leq \epsilon$. To this end, we will first construct a probability distribution D^* on $X^n \times [n] \times Y$. By the easier direction of the minimax lemma, we will get a $[t, c, a, b]^A$ safe coinless protocol P^* for $f^{(n)}$ with distributional error, for distribution D^*, $\epsilon_{D^*}^{P^*} \leq \delta$. We shall construct the desired protocol P from the protocol P^*.

The distribution D^* is constructed as follows. Choose $i \in [n]$ uniformly at random. Choose independently, for each $j \in [n]$, $(x_j, y_j) \in X \times Y$ according to distribution D. Set $y = y_i$ and throw away $y_j, j \neq i$.

Let M be the first message of Alice in P^*. By the definition of a safe protocol, M has two parts, M_1 a qubits long, and the "safe" overhead M_2, c qubits long. Let the input to Alice be denoted by the classical random variable $X = X_1 X_2 \ldots X_n$ where X_i is the classical random variable corresponding

to the ith input to Alice. Define $\epsilon^{P^*}_{\substack{D^*,i \\ x_1,\ldots,x_{i-1}}}$ to be the average error of P^* under distribution D^* when i is fixed and $X_1,\ldots,X_{i-1}$ are fixed to $x_1,\ldots,x_{i-1}$. From Propositions 1, 2, 3, using the fact that under distribution D^* $X_1,\ldots,X_n$ are independent classical random variables, we get that

$$\begin{aligned} 2a/n &\geq \frac{I(X:M)}{n} \\ &= E_i[I(X_i : MX_1,\ldots,X_{i-1})] \\ &= E_{i,X}[I((X_i:M)|X_1,\ldots,X_{i-1}=x_1,\ldots,x_{i-1})] \end{aligned}$$

Also

$$\delta \geq \epsilon^{P^*}_{D^*} = E_{i,X}\left[\epsilon^{P^*}_{\substack{D^*,i \\ x_1,\ldots,x_{i-1}}}\right]$$

By two applications of Markov's inequality, we see that there exists a choice of i and $x_1,\ldots,x_{i-1}$ such that, if we define a new distribution $\tilde{D}$ to be distribution D^* where i is fixed to the above choice and $X_1,\ldots,X_{i-1}$ are fixed to $x_1,\ldots,x_{i-1}$, then the error of the protocol P^* on distribution $\tilde{D}$ $\epsilon^{P^*}_{\tilde{D}} \leq 2\delta$ and the mutual information between X_i and M under distribution $\tilde{D}$ $I_{\tilde{D}}(X_i : M) \leq 4a/n$.

Consider now the protocol P' for the function f defined as follows. P' is a $[t,c,a,b]^A$ safe coinless quantum protocol. Alice is given $x \in X$ and Bob is given $y \in Y$. Both Alice and Bob set i to the above choice (which is known to both parties) and $X_1 \ldots X_{i-1}$ to the known values $x_1 \ldots x_{i-1}$. Alice puts an independent copy of a pure state $|\psi\rangle$ for each of the inputs $X_{i+1},\ldots,X_n$. She sets $X_i = x$ and Bob sets his input $Y = y$. Then they run protocol P^* on these inputs. Here $|\psi\rangle \stackrel{\Delta}{=} \sum_{x\in X}\sqrt{p_x}|x\rangle$, where p_x is the (marginal) probability of x under distribution D. Since P^* is a secure protocol, the probability that P' makes an error for an input (x,y), $\epsilon^{P'}_{x,y}$, is the average probability of error of P^* under distribution $\tilde{D}$ with X_i, Y fixed to x, y. Hence, the average probability of error of P' under distribution D $\epsilon^{P'}_{D} = \epsilon^{P^*}_{\tilde{D}} \leq 2\delta$. Also, because of the "secureness" of P^*, we notice that the mutual information between X_i and the first message of P' (under distribution D) is the same as the mutual information between X_i and the first message of P^* (under distribution $\tilde{D}$). Thus, if M denotes the first message of P' and X denotes the register X_i holding the input x, then the mutual information under distribution D, $I_D(X : M) \leq 4a/n$.

Since in protocol P' the first message of Alice has small mutual information with her input, we can give an argument similar to Nayak et al. [10], and finally get a $[t-1, c+a, a, b]^B$ safe coinless quantum protocol P for f with $\epsilon^P_D \leq \epsilon^{P'}_D + 2((2\ln 2)I_D(X:M))^{1/4} \leq 2\delta + 2(8a\ln 2/n)^{1/4} = \epsilon$. For this we have to use the version of the "average encoding theorem" as in Theorem 1, instead of the version of [10], which held for uniform probability distributions only. We observe, in the construction of P from P', that though there is a overhead of $a+c$ qubits on the first message of Bob, it is a "safe" overhead. The details are left to the full version [14].

This completes the proof of the round elimination lemma. □

From this lemma, we can prove the round elimination lemma in the form it will be used in various applications.

Lemma 3 (Round elimination lemma for fixed error). *Suppose $f: X \times Y \to Z$ is a function. There exist universal constants R, C such that the following holds: Suppose that the communication game $f^{(Ra)}$ has a $[t, c, a, b]^A$ safe public coin quantum protocol with error probability at most $1/3$. Then the communication game f has a $[t-1, C(a+c), Ca, Cb]^B$ safe public coin quantum protocol with error probability at most $1/3$. For example, $R = 10^4$ and $C = 51$ suffices.*

Proof. (**Sketch**) Repeat the $[t, c, a, b]^A$ protocol for $f^{(Ra)}$ C times in parallel and take the majority of the results. This brings the error probability down to a suitably small value. Now apply Lemma 2 on the repeated protocol. □

5 Quantum Lower Bounds for Predecessor

In this section, we prove our lower bounds for the static predecessor problem in the address-only quantum cell probe model. The proof is essentially similar to the classical proof in Miltersen et al. [9], and hence we give only a brief sketch.

Theorem 2. *Suppose we have a $(n^{O(1)}, O(\log m), t)$ quantum address-only cell probe solution to the static predecessor problem, where the universe size is m and the subset size is at most n. Then the number of queries t is at least $\Omega(\log^{1/3} n)$ as a function of n, and it is at least $\Omega(\sqrt{\log \log m})$ as a function of m.*

Proof. (**Sketch**) We basically imitate the proof of Miltersen et al [9], but in our quantum setting. By Lemma 1, it suffices to prove a lower bound on the number of rounds of a communication game. For that, we alternately use "self-reducibility" arguments and the round elimination lemma (Lemma 3) to keep reducing the number of rounds in the communication game. One just has to notice that the applicability of Lemma 3 does not depend on the "safe" overhead at all, but rather on the per round message complexity of the first player. This allows the quantum arguments to go through in a manner similar to the classical arguments, and hence, proves our theorem. □

Miltersen et al. also apply the round elimination lemma to prove lower bounds for other data structure problems and communication complexity problems. We remark that we can extend all those results in a similar fashion to the quantum world.

Acknowledgements. We thank Ashwin Nayak, Jaikumar Radhakrishnan and Rahul Jain for useful discussions, Hartmut Klauck for his clarifications on the average encoding theorem, and Peter Bro Miltersen for telling us the "state-of-the-art" about the classical complexity of the static predecessor problem. We also thank Jaikumar Radhakrishnan for reading an early draft of this paper and helping us to improve the presentation of this paper.

References

[1] P. Beame and F. Fich. Optimal bounds for the predecessor problem. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 295–304, 1999.

[2] H. Buhrman, P. Miltersen, J. Radhakrishnan, and S. Venkatesh. Are bitvectors optimal? In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 449–458, 2000.

[3] R. Cleve, W. van Dam, , M. Nielsen, and A. Tapp. Quantum entanglement and the communication complexity of the inner product function. In *Proceedings of the 1st NASA International Conference on Quantum Computing and Quantum Communications*, Lecture Notes in Computer Science, vol. 1509, pages 61–74, 1998. Also quant-ph/9708019.

[4] M. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the Association for Computing Machinery*, 31(3):538–544, 1984.

[5] L. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996. Also quant-ph/9605043.

[6] P. Høyer and J. Neerbek. Bounds on quantum ordered searching. Manuscript at quant-ph/0009032, September 2000.

[7] H. Klauck. On rounds in quantum communication. Manuscript at quant-ph/0004100, April 2000.

[8] P. B. Miltersen. Cell probe complexity — a survey. Invited talk at the pre-conference workshop on *Advances in data structures preceding the 19th conference on the Foundations of Software Technology and Theoretical Computer Science,* December 11–12, 1999, Chennai, India. Also available from http://www.daimi.au.dk/~bromille/Papers/survey3.ps.

[9] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57:37–49, 1998.

[10] A. Nayak, A. Ta-Shma, and D. Zuckerman. Interaction in quantum communication complexity. Manuscript at quant-ph/0005106, May 2000.

[11] I. Newman. Private vs common random bits in communication complexity. *Information Processing Letters*, 39:67–71, 1991.

[12] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.

[13] J. Radhakrishnan, P. Sen, and S. Venkatesh. The quantum complexity of set membership. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 554–562, 2000. Also quant-ph/0007021.

[14] P. Sen and S. Venkatesh. Lower bounds in the quantum cell probe model. Full version. Manuscript at quant-ph/0104100.

[15] A. C-C. Yao. Probabilistic computations: towards a unified measure of complexity. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, pages 209–213, 1977.

[16] A. C-C. Yao. Should tables be sorted? *Journal of the Association for Computing Machinery*, 28(3):615–628, 1981.

[17] A. C-C. Yao. Quantum circuit complexity. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, pages 352–361, 1993.

Axiomatizations for Probabilistic Bisimulation*

Emanuele Bandini and Roberto Segala

Department of Computer Science
Università di Bologna - Italy

Abstract. We study complete axiomatizations for different notions of probabilistic bisimulation on a recursion free process algebra with probability and nondeterminism under alternating and non-alternating semantics. The axioms that do not involve probability coincide with the original axioms of Milner. The axioms that involve probability differ depending on the bisimulation under examination and on the semantics that is used, thus revealing the implications of the different choices.

1 Introduction

Probabilistic process algebras have been studied extensively in the literature [1,3, 4,6,8,13,16], and classical concepts from concurrency theory have been extended to the probabilistic case. Probabilistic models of concurrent systems are classified in [5] into *reactive*, *generative*, and *stratified*. Both in reactive and generative systems the transitions that leave from a state are equipped with probabilities: in generative systems the sum of the probabilities of the transitions that leave from a state is required to be 1, while in reactive systems the sums of the probabilities of the transitions that leave from a state and are labeled by the same action are required to be 1. The stratified model imposes some extra structure which is not relevant for the purpose of this paper.

Motivated by the fact that neither reactive nor generative nor stratified systems model real nondeterminism in the process algebraic sense, and motivated as well by the desire to separate clearly probability from nondeterminism, in [11] a model of *probabilistic automata* is introduced and studied. Probabilistic automata, and more precisely the simple probabilistic automata of [11], are like ordinary automata (labeled transition systems) except that a transition leads to a probability distribution over states rather than to a single state. Thus, the choice between different transitions is a nondeterministic choice, while the choice of a state within a transition is a probabilistic choice. A similar model was proposed in [6] based on the Concurrent Markov Chains of [15]. In such model, also known as the *alternating model*, there is a clear distinction between *nondeterministic states*, that enable only transitions leading to a unique state, and *probabilistic states*, that enable a unique transition leading to a distribution over states. There is a strict alternation between nondeterministic and probabilistic

* Supported by MURST project TOSCA.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 370–381, 2001.

states. Both the alternating model and the model of [11], which in contraposition to the alternating model is also known as the *non-alternating model*, are conservative extensions of labeled transition systems and in several contexts can be seen as the same model from the point of view of expressiveness.

Yet, the alternating and non-alternating models do have some differences that can be seen already when we study bisimulation relations. Probabilistic bisimulation was first defined in [8], then extended to the alternating model in [6] and extended to the non-alternating model in [12]. While defining probabilistic bisimulation in the non-alternating model it was shown in [12] that we obtain two different relations if we simulate a transition using deterministic and randomized schedulers, respectively. Such difference does not appear in the alternating model unless we change the definitions of probabilistic bisimulations so that probabilistic states are not taken into account.

In this paper we show the differences and similarities of the alternating and non-alternating models by analyzing the axiomatizations of the different bisimulation relations in the different frameworks. We define a process algebra without recursion and provide it with an alternating and non-alternating semantics. We then define a *strong bisimulation* relation that coincides with the relation of [6] in the alternating model and with the bisimulation of [12] in the non-alternating model. We also define the version of strong bisimulation, called *strong probabilistic bisimulation*, where a transition can be simulated by using randomized schedulers. Finally, we study the complete axiomatizations of all the relations that we introduce. Besides obtaining axioms where probability and nondeterminism are separated clearly, thus confirming the original goal behind the definitions of the models, we discover that the axiomatizations of strong bisimulation are the same in the alternating and non-alternating models. Furthermore, the axiomatizations of strong bisimulation and strong probabilistic bisimulation are the same in the alternating model, while they differ by an axiom that expresses the ability to combine transitions probabilistically in the non-alternating model.

We also study the weak bisimulations of [12], showing that the alternating and non-alternating semantics are incomparable.

Other studies of axiomatizations for probabilistic bisimulation relations appear in [2,6,7,14]. Of these axiomatizations, only [6] deals with a reactive model. The axiomatization of [6] includes recursion as well.

The rest of the paper is structured as follows. Section 2 gives some preliminary definitions and notational conventions; Section 3 defines the Probabilistic Process Algebra (PPA) and its alternating and non-alternating semantics; Section 4 defines the bisimulation relations that we axiomatize; Section 5 axiomatizes the relations of Section 4, discusses the axioms, and outlines the main ideas behind the proofs of completeness; Section 6 contains some concluding remarks.

2 Preliminaries

A *discrete probability measure* over a set X is a function $\mu : 2^X \to [0,1]$ such that $\mu(X) = 1$ and for each countable family $\{X_i\}$ of pairwise disjoint elements

of 2^X, $\mu(\cup_i X_i) = \sum_i \mu(X_i)$. Denote by $Disc(X)$ the set of discrete probability measures over X. Given an element x of X we denote by $\delta(x)$ the probability measure μ such that $\mu(\{x\}) = 1$, and we call it the *Dirac measure* on x. Given two measures μ_1, μ_2 and a real number $p \in [0,1]$ we define the *convex combination* $p\mu_1 + (1-p)\mu_2$ of μ_1 and μ_2 to be the probability measure μ such that, for each set Y, $\mu(Y) = p\mu_1(Y) + (1-p)\mu_2(Y)$.

A *probabilistic automaton* is a tuple $(Q, \bar{q}, \Sigma, \mathcal{D})$, where Q is a set of *states*, $\bar{q} \in Q$ is a *start state*, Σ is a set of *actions*, and $\mathcal{D} \subseteq Q \times \Sigma \times Disc(Q)$ is a *transition relation*. An ordinary automaton can be seen as a probabilistic automaton where each transition leads to a Dirac measure. Probabilistic automata are used as the basis to give an operational semantics to our probabilistic process algebra.

3 Probabilistic Process Algebra

We denote by $\mathcal{A}$ the set of observable *actions* or *labels*, and let $Act = \mathcal{L} \cup \{\tau\}$ be the full set of actions. We call τ the *silent action* and we let α range over Act.

Let *NProc* denote the set of *nondeterministic processes*, ranged over by E, and *PProc* denote the set of *probabilistic processes*, ranged over by P. Finally, let $Proc \triangleq NProc \cup PProc$ denote the set of *processes*, ranged over by Q. The syntax for our Probabilistic Process Algebra is given by the following rules:

$$E ::= 0 \mid E + E \mid \alpha.P$$
$$P ::= \Delta(E) \mid P \oplus_p P$$

The expression 0 is the inactive process having no transitions. The $+$ operator is the classical nondeterministic sum as defined in [9]. Process $\alpha.P$ performs action α and then offers a probabilistic choice described by the probabilistic process P. A probabilistic process is either a Dirac distribution over a single nondeterministic process, described by $\Delta(E)$, or a combination of the distributions associated with two probabilistic processes, described by the $\oplus_p$ operator.

For notational convenience we can represent sums of nondeterministic processes by $\sum_{i \in I} E_i$ and sums of probabilistic processes by $\sum_{i \in I} [p_i] E_i$. Such representations are justified by the fact that in this paper both the operators $+$ and $\oplus_p$ turn out to be associative and commutative. We let μ range over distributions over nondeterministic processes and sometimes we represent a distribution over nondeterministic processes by $\{[p_i]E_i\}_{i \in I}$.

Note that PPA is characterized by a strict alternation between probabilistic and nondeterministic processes as in [6]. The alternation is kept in the alternating semantics of the calculus and is removed in the non-alternating semantics.

Table 1 contains the operational semantics of PPA, where $E \xrightarrow{\alpha} \mu$ describes a transition labeled by α that leaves from E and leads to a probability distribution μ, while $P \longmapsto \mu$ states that the probability distribution associated with P is μ. The rules of Table 1 describe the transitions of a probabilistic automaton; thus, the target of a transition of Table 1 is a probability distribution over expressions rather than a single expression.

Table 1. Operational semantics of PPA

Probabilistic rules

$$\textbf{idle}\quad \frac{-}{\Delta(E) \longmapsto \delta(E)} \qquad \textbf{pchoice}\quad \frac{P_1 \longmapsto \mu_1 \qquad P_2 \longmapsto \mu_2}{P_1 \oplus_p P_2 \longmapsto p\mu_1 + (1-p)\mu_2}$$

Common nondeterministic rules

$$\textbf{lchoice}\quad \frac{E_1 \stackrel{\alpha}{\longrightarrow} \mu}{E_1 + E_2 \stackrel{\alpha}{\longrightarrow} \mu} \qquad \textbf{rchoice}\quad \frac{E_2 \stackrel{\alpha}{\longrightarrow} \mu}{E_1 + E_2 \stackrel{\alpha}{\longrightarrow} \mu}$$

$$\textbf{P}-\textbf{idle}\quad \frac{P \longmapsto \mu}{P \stackrel{\tau}{\longrightarrow} \mu}$$

Rule for non alternating model

$$\textbf{NA}-\textbf{prefix}\quad \frac{P \longmapsto \mu}{\alpha.P \stackrel{\alpha}{\longrightarrow} \mu}$$

Rule for alternating model

$$\textbf{A}-\textbf{prefix}\quad \frac{-}{\alpha.P \stackrel{\alpha}{\longrightarrow} \delta(P)}$$

Table 1 is subdivided into three sections. The first section defines the probability distributions associated with a probabilistic process. Specifically, process $\Delta(E)$ is associated with a Dirac distribution over the single process E (rule **idle**), while the probability distribution associated with the probabilistic combination $P_1 \oplus_p P_2$ is obtained by convex combination weighted by p of the distributions associated with P_1 and P_2, respectively (rule **pchoice**). The second section of Table 1 describes the operators whose semantics does not change in the alternating and non-alternating interpretations. Specifically, the semantics of the $+$ operator is the same as in CCS (rules **lchoice** and **rchoice**). Rule **P-idle** describes the unique transition that is enabled from a probabilistic process, which moves silently to the distribution associated with the process. This rule is essential in the alternating semantics, where probabilistic processes can be reached; however, the same rule is convenient also in the non-alternating semantics to obtain an axiomatization of probabilistic bisimulation that reveals better the relationship between the two semantics. The third section of Table 1 contains the rules for action-prefixing, which constitute the key difference between the alternating and non-alternating semantics. In the non-alternating semantics process $\alpha.P$ moves with action α to the distribution identified by P (rule **NA-prefix**), while in the alternating semantics process $\alpha.P$ moves with action α to process

P (rule **A-prefix**) from which a silent move leads to the distribution identified by P (cf. rule **P-idle**).

Remark 1. There is a folklore idea of how an alternating system can be translated into a non-alternating system and vice versa. Specifically, to move from an alternating system to a non-alternating system it is sufficient to remove all the probabilistic states and collapse the transitions that go through a probabilistic state, while to move from a non-alternating system to an alternating system it is sufficient to split each transition into two transitions, the first of which leads to a probabilistic state. The operational semantics of Table 1 respects the folklore transformation: for each process E the transformation of its alternating semantics coincides with its non-alternating semantics and vice versa.

4 Bisimulation

In this section we define bisimulation relations in the strong and weak version based on deterministic and randomized schedulers. In the non-alternating model our definition of strong and weak (probabilistic) bisimulation coincide with those of [12]; in the alternating model strong bisimulation coincides with the strong bisimulation of [6], while weak probabilistic bisimulation coincides with the weak bisimulation of [10].

4.1 Lifting Equivalence Relations

An equivalence relation over *Proc* can be lifted to a relation over distributions over *Proc* by stating that two distributions are equivalent if they assign the same probability to the same equivalence classes [8].

Formally, let $\mathcal{R}$ be an equivalence relation over *Proc*. Two probability distributions μ_1 and μ_2 are $\mathcal{R}$-equivalent, written $\mu \; \mathcal{R}_p \; \mu'$, iff for every equivalence class $\mathcal{E} \in \mathit{Proc}/\mathcal{R}$ we have $\mu(\mathcal{E}) = \mu'(\mathcal{E})$.

4.2 Strong Bisimulation

An equivalence relation $\mathcal{R} \subseteq \mathit{Proc} \times \mathit{Proc}$ is a *strong bisimulation* iff, for all $Q_1, Q_2 \in \mathit{Proc}$ such that $Q_1 \; \mathcal{R} \; Q_2$, and for all $\alpha \in \mathit{Act}$,

- if $Q_1 \xrightarrow{\alpha} \mu_1$, then there exists μ_2 such that $Q_2 \xrightarrow{\alpha} \mu_2$ and $\mu_1 \; \mathcal{R}_p \; \mu_2$;
- if $Q_2 \xrightarrow{\alpha} \mu_2$, then there exists μ_1 such that $Q_1 \xrightarrow{\alpha} \mu_1$ and $\mu_1 \; \mathcal{R}_p \; \mu_2$.

We write $Q_1 \sim Q_2$ whenever there is a strong bisimulation that relates Q_1, Q_2.

Proposition 1. *Strong bisimulation is a congruence in PPA.*

In a strong bisimulation a transition of a process must be simulated by a single transition of the other process chosen deterministically among the transitions that are enabled. It was observed in [12] that deterministic schedulers may not be enough in a randomized setting.

Example 1. Consider $E \triangleq \alpha.(\Delta(E_1) \oplus_{1/2} \Delta(E_2)) + \alpha.(\Delta(E_1) \oplus_{1/3} \Delta(E_2))$ and $F \triangleq \alpha.(\Delta(E_1) \oplus_{1/2} \Delta(E_2)) + \alpha.(\Delta(E_1) \oplus_{5/12} \Delta(E_2)) + \alpha.(\Delta(E_1) \oplus_{1/3} \Delta(E_2))$ whose non-alternating semantics is represented in Figure 1. Each bundle of edges corresponds to a transition. The difference between E and F is that F enables

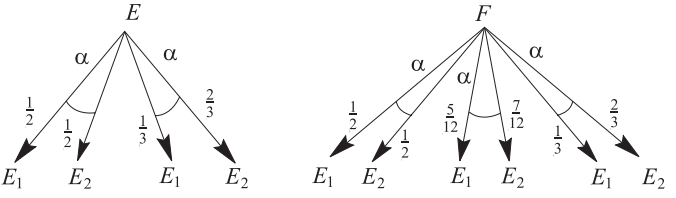


Fig. 1. Two processes not strongly bisimilar

an additional transition which is obtained by combining probabilistically the two transitions of E. There is no strong bisimulation between E and F if E_1 and E_2 are not bisimilar; however, E and F would be bisimilar if we permit the use of randomized schedulers to simulate the extra transition of F.

Example 1 suggests a new bisimulation relation where it is possible to combine several transitions labeled by the same action in a unique transition. We say that there is a *combined transition* labeled by action α from a process E to a distribution μ, denoted by $E \xrightarrow{\alpha}_C \mu$, iff there exists a collection $\{\mu_i, p_i\}_{i \in I}$ of distributions and probabilities such that $\sum p_i = 1$, $\mu = \sum p_i \mu_i$, and $\forall i : E \xrightarrow{\alpha} \mu_i$.

An equivalence relation $\mathcal{R} \subseteq Proc \times Proc$ is a *strong probabilistic bisimulation* iff, for all $Q_1, Q_2 \in Proc$ such that $Q_1 \mathcal{R} Q_2$, and for all $\alpha \in Act$,

- if $Q_1 \xrightarrow{\alpha} \mu_1$, then there exists μ_2 such that $Q_2 \xrightarrow{\alpha}_C \mu_2$ and $\mu_1 \mathcal{R}_p \mu_2$;
- if $Q_2 \xrightarrow{\alpha} \mu_2$, then there exists μ_1 such that $Q_1 \xrightarrow{\alpha}_C \mu_1$ and $\mu_1 \mathcal{R}_p \mu_2$.

We write $Q_1 \sim_C Q_2$ whenever there is a strong probabilistic bisimulation that relates Q_1 and Q_2.

Proposition 2. *Strong probabilistic bisimulation is a congruence in PPA.*

It is easy to observe that strong bisimulation is just a particular case of strong probabilistic bisimulation. An important result is that in the alternating semantics strong bisimulation coincides with strong probabilistic bisimulation (cf. Proposition 3). Thus, randomized schedulers do not add any extra power to the ability of simulating a transition. Roughly speaking, in the alternating model each probability distribution is declared explicitly through a probabilistic state before being drawn. Strong bisimulation must preserve the declarations as well, and on the other hand there is no way to declare the combination of two transitions.

Proposition 3. *Under the alternating semantics a strong probabilistic bisimulation is also a strong bisimulation.*

Table 2. Weak transitions

$$\frac{E \xrightarrow{\alpha} \mu}{E \stackrel{\alpha}{\Longrightarrow} \mu} \qquad \frac{E \xrightarrow{\tau} \mu}{E \Longrightarrow \mu} \qquad \frac{-}{E \Longrightarrow \delta(E)}$$

$$\frac{E \stackrel{\alpha}{\Longrightarrow} \mu \qquad \forall_{E_i \in \mu} E_i \Longrightarrow \mu_i}{E \stackrel{\alpha}{\Longrightarrow} \sum_{E_i \in \mu} \mu(E_i)\mu_i} \qquad \frac{E \Longrightarrow \mu \qquad \forall_{E_i \in \mu} E_i \stackrel{\alpha}{\Longrightarrow} \mu_i}{E \stackrel{\alpha}{\Longrightarrow} \sum_{E_i \in \mu} \mu(E_i)\mu_i}$$

Proof sketch. Let $\sim_C$ be a strong probabilistic bisimulation and suppose $Q_1 \sim_C Q_2$. If Q_1 and Q_2 are probabilistic processes, then they enable only one transition, the silent transition that selects probabilistically one process. Thus, there is nothing to combine. If Q_1 and Q_2 are nondeterministic processes and $Q_1 \xrightarrow{\alpha} \mu$, then μ is a Dirac distribution over some probabilistic process P. The combined transition $Q_2 \xrightarrow{\alpha}_C \mu'$ that simulates $Q_1 \xrightarrow{\alpha} \mu$ leads to a distribution that assigns probability 1 to the equivalence class of P. Thus, any transition from Q_2 that contributes to $Q_2 \xrightarrow{\alpha}_C \mu'$ leads to distribution that assigns probability 1 to the equivalence class of P. This shows that $Q_1 \sim Q_2$.

4.3 Weak Bisimulation

Weak bisimulation is the same as strong bisimulation except that we replace transitions by weak transitions. That is, we are not interested in observing the silent behavior of a system. A weak transition, whose formal definition is given in Table 2, is a probabilistic extension of the weak transitions of [9]. We schedule several transitions as long as they always lead to the occurrence of a single external action α, possibly interleaved by silent actions. For notational convenience, given a sequence s of actions in Act, we denote by $\widehat{s}$ the sequence obtained from s by removing all τ's.

An equivalence relation $\mathcal{R} \subseteq Proc \times Proc$ is a *weak bisimulation* iff, for all $Q_1, Q_2 \in Proc$ such that $Q_1 \mathcal{R} Q_2$, and for all $\alpha \in Act$,

- if $Q_1 \xrightarrow{\alpha} \mu_1$ then there exists μ_2 such that $Q_2 \stackrel{\widehat{\alpha}}{\Longrightarrow} \mu_2$ and $\mu_1 \mathcal{R}_p \mu_2$;
- if $Q_2 \xrightarrow{\alpha} \mu_2$ then there exists μ_1 such that $Q_1 \stackrel{\widehat{\alpha}}{\Longrightarrow} \mu_1$ and $\mu_1 \mathcal{R}_p \mu_2$.

We write $Q_1 \approx Q_2$ whenever there is a weak bisimulation that relates Q_1 and Q_2.

We can define a *weak combined transition relation* ($\Longrightarrow_C$), as we have done in the strong case, by combining simple weak transitions. Thus, it is possible to define *weak probabilistic bisimulation* by replacing weak transitions by weak combined transitions in the definition above.

4.4 Observation Congruence

As in ordinary CCS [9], weak bisimulation is not preserved by the nondeterministic choice operator $+$. The classical example is given by the pair of processes $a.0 \approx \tau.a.0$, which are equivalent both according to weak bisimulation and weak probabilistic bisimulation, where $a.0 + b.0 \not\approx \tau.a.0 + b.0$. Following the classical approach of [9], we define *observation congruence* and *probabilistic observation congruence*.

Two processes Q_1, Q_2 are *congruent*, written $Q_1 = Q_2$, if Q_1 and Q_2 are both nondeterministic or both probabilistic, and for all $\alpha \in Act$,

- if $Q_1 \xrightarrow{\alpha} \mu_1$ then there exists μ_2 such that $Q_2 \stackrel{\alpha}{\Longrightarrow} \mu_2$ and $\mu_1 \approx \mu_2$
- if $Q_2 \xrightarrow{\alpha} \mu_2$ then there exists μ_1 such that $Q_1 \stackrel{\alpha}{\Longrightarrow} \mu_1$ and $\mu_1 \approx \mu_2$

Two processes Q_1, Q_2 are *probabilistically congruent*, written $Q_1 =_C Q_2$, if Q_1 and Q_2 are both nondeterministic or both probabilistic, and for all $\alpha \in Act$,

- if $Q_1 \xrightarrow{\alpha} \mu_1$ then there exists μ_2 such that $Q_2 \stackrel{\alpha}{\Longrightarrow} \mu_2$ and $\mu_1 \approx_C \mu_2$
- if $Q_2 \xrightarrow{\alpha} \mu_2$ then there exists μ_1 such that $Q_1 \stackrel{\alpha}{\Longrightarrow} \mu_1$ and $\mu_1 \approx_C \mu_2$

The only difference between congruence and weak bisimulation is that in the former there is $\stackrel{\alpha}{\Longrightarrow}$ instead of $\stackrel{\widehat{\alpha}}{\Longrightarrow}$. This implies that every τ-transition of Q_1 is related with *at least* one τ-transition of Q_2, and vice versa. Observe that this strong relationship is requested only for the first transitions of both Q_1 and Q_2: in fact, it is sufficient that $\mu_1 \approx_p \mu_2$, not $\mu_1 =_p \mu_2$.

Proposition 4. *The relations $=$ and $=_C$ are congruences in PPA.*

5 Axiomatizations

5.1 Discussion of the Axioms

The axioms that characterize completely the bisimulation relations of this paper are listed in Table 3. The left side of Table 3 contains the axioms for the non-alternating semantics of PPA, while the right part contains the axioms for the alternating semantics of PPA. Table 3 is also subdivided into four horizontal sections. The first and third sections axiomatize strong bisimulation. By adding the second section we obtain complete axiomatizations for observation congruence. Finally, by adding the fourth section we obtain complete axiomatizations for the probabilistic versions of our bisimulations, where axiom **CW** holds only for the weak relations. Thus, sections 1, 3 and 4 provide complete axiomatizations for the strong probabilistic bisimulations.

Observe that there is no **C** axiom in the right column of Table 3, which confirms that strong bisimulation is the same under randomized and non-randomized schedulers in the alternating semantics. Furthermore, there is no **CW** axiom in the left column of Table 3, which shows that randomization adds some restricted power to the ability of simulating a weak transition in the alternating model. Axiom **CW** does not hold in the non-alternating semantics since

the term $\tau.P$ reached after the α-labeled transition of $\alpha.(P + \Delta(\tau.P))$ cannot be simulated in general by the distribution identified by P in $\alpha.P$. See also the discussion about axiom **A8**.

Observe that the first and third sections of Table 3 contain the same axioms in the two columns. This confirms that under strong bisimulation with deterministic schedulers the alternating and non-alternating models are indeed the same. We can observe a difference between alternating and non-alternating semantics in the second section of Table 3. Specifically, axioms **A6-7** of the right column are more restrictive than the axioms of the left column (P_i replaced by P). On the other hand, axiom **A8** holds only in the alternating semantics, thus showing that weak bisimulations are incomparable. Axiom **A8** expresses the informal idea that in the alternating model each distribution must be declared before being drawn. Thus, adding further declarations does not matter. The left version of axiom **A5** can be replaced by its right version. We have kept both versions to illustrate better the analogies with the τ-laws of Milner.

Another important observation is that the axiomatizations of Table 3 keep most of the structure of the axiomatizations for ordinary CCS [9]. The axioms of the first section are exactly the axioms for strong bisimulation on CCS, and the axioms of the third sections add the ingredients that are need for the new probabilistic choice operator. The τ-laws of the second section have the same structure of the τ-laws of Milner, except that within a prefix we have the probabilistic choice operator. If we consider processes without the probabilistic choice operator, then our τ-laws coincide with the τ-laws of Milner.

5.2 Proof Sketches

The proofs of the completeness results are similar to the corresponding proofs for CCS [9]: a process is reduced to a normal form, possibly saturated, and then processes are compared almost syntactically piece by piece. In this section we give an overview of the normal forms that are needed in the proofs.

Definition 1. *A nondeterministic process E is in* normal form *(NF) if*

$$E \equiv \sum_{i \in I} \alpha_i . \sum_{j \in J_i} [p_j^i] E_j^i$$

where the processes E_j^i are in normal form as well.

Getting a process in normal form is almost immediate since it is sufficient to remove all exceeding 0's by using axiom **A4** and the congruence rules.

Definition 2. *A nondeterministic process E is in* strict normal form *(SNF) if*

$$E \equiv \sum_{i \in I} \alpha_i . \sum_{j \in J_i} [p_j^i] E_j^i$$

where $\forall_i \forall_{j,j' \in J_i}$ if $\mathcal{S} \vdash E_j^i = E_{j'}^i$, then $j = j'$. With $\mathcal{S}$ we denote the axioms of the first and third sections of Table 3.

Table 3. Axioms for strong and weak bisimulations

	Non alternating semantics	**Alternating semantics**
A1	$E + F = F + E$	$E + F = F + E$
A2	$E + (F + G) = (E + F) + G$	$E + (F + G) = (E + F) + G$
A3	$E + E = E$	$E + E = E$
A4	$E + 0 = E$	$E + 0 = E$
A5	$\alpha.(\Delta(\tau.\Delta(E)) \oplus_p P) = \alpha.(\Delta(E) \oplus_p P)$	$\Delta(\tau.\Delta(E)) = \Delta(E)$
A6	$\tau.\sum_{i\in I}[p_i](E_i + \alpha.P_i) + \alpha.\sum_{i\in I}[p_i]P_i = \tau.\sum_{i\in I}[p_i](E_i + \alpha.P_i)$	$\tau.\sum_{i\in I}[p_i](E_i + \alpha.P) + \alpha.P = \tau.\sum_{i\in I}[p_i](E_i + \alpha.P)$
A7	$\alpha.\sum_{i\in I}[p_i](E_i + \tau.P_i) + \alpha.\sum_{i\in I}[p_i]P_i = \alpha.\sum_{i\in I}[p_i](E_i + \tau.P_i)$	$\alpha.\sum_{i\in I}[p_i](E_i + \tau.P) + \alpha.P = \alpha.\sum_{i\in I}[p_i](E_i + \tau.P)$
A8	-	$\alpha.P = \alpha.\Delta(\tau.P)$
P1	$P \oplus_p Q = Q \oplus_{(1-p)} P$	$P \oplus_p Q = Q \oplus_{(1-p)} P$
P2	$P \oplus_{p_1} (Q \oplus_{\frac{p_2}{1-p_1}} R) = (P \oplus_{\frac{p_1}{p_1+p_2}} Q) \oplus_{(p_1+p_2)} R$	$P \oplus_{p_1} (Q \oplus_{\frac{p_2}{1-p_1}} R) = (P \oplus_{\frac{p_1}{p_1+p_2}} Q) \oplus_{(p_1+p_2)} R$
P3	$P \oplus_p P = P$	$P \oplus_p P = P$
C	$\alpha.P_1 + \alpha.P_2 = \alpha.P_1 + \alpha.P_2 + \alpha.(P_1 \oplus_p P_2)$	-
CW	-	$\alpha.(P \oplus \Delta(\tau.P)) = \alpha.P$

To get a process in strict normal form we first convert the process to normal form. Then, whenever we find two elements E^i_j and $E^i_{j'}$ that are provably equivalent, we use axiom **P3** to collapse them. Of course we need also axioms **P1** and **P2** to get the two terms next to each other.

Processes in strict normal form are sufficient for the proof of completeness for strong bisimulation that works prefix by prefix as in [9]. To handle strong probabilistic bisimulation we use axiom **C** to build the missing summands that originate from convex combinations of other summands. Thus, we reduce strong probabilistic bisimulation to strong bisimulation.

To deal with weak bisimulation we need to saturate a process as in [9]. For this purpose we define complete normal forms.

Definition 3. *A nondeterministic process E is in* complete normal form *(CNF) if*

- $E \equiv \sum_{i \in I} \alpha_i . \sum_{j \in J_i} [p_j^i] E_j^i$
- *the processes E_j^i are in CNF*
- *if $E \stackrel{\alpha}{\Longrightarrow} \mu$, then $E \stackrel{\alpha}{\longrightarrow} \mu$.*

The saturation process to get an expression in complete normal form consists of using axiom **A6** to move out of a τ-prefix each transition labeled by some external action. The final step is to get a strict complete normal form in the same way as we do for strong bisimulation. When axiomatizing weak probabilistic bisimulation, once again we use axiom **C** to create the missing summands.

The normal form for weak bisimulation in the alternating semantics differs from the normal form in the non-alternating semantics in that axiom **A6** allows us to saturate only those transitions that lead to Dirac distributions.

Definition 4. *A nondeterministic process E is in* alternating complete normal form *(ACNF) if*

- $E \equiv \sum_{i \in I} \alpha_i . \sum_{j \in J_i} [p_j^i] E_j^i$
- *the processes E_j^i are in ACNF*
- *if $E \stackrel{\alpha}{\Longrightarrow} \delta(P)$, then $E \stackrel{\alpha}{\longrightarrow} \delta(P)$.*

6 Concluding Remarks

We have studied axiomatizations of bisimulation relations for a recursion free fragment of a probabilistic process algebra that includes probabilistic and nondeterministic choices. Our analysis included strong and weak bisimulation, deterministic and randomized schedulers, alternating and non-alternating semantics.

The axioms have a structure consistent with the original axioms of Milner and separate clearly the concerns of nondeterminism and probability. The axiomatizations that we have found also highlight the main differences and similarities of the alternating and non-alternating models of concurrent probabilistic systems.

We are currently planning to extend our axiomatizations to a probabilistic process algebra with recursion and parallel composition. We do not expect any special surprises with parallel composition since a probabilistic generalization of the expansion law of Milner is easy to derive.

References

1. S. Andova. Process algebra with probabilistic choice. In *Formal Methods for Real-Time and Probabilistic Systems*, LNCS 1601, pages 111–129, 1999.
2. J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation*, 122:234–255, 1995.
3. M. Bernardo, L. Donatiello, and R. Gorrieri. Modeling and analyzing concurrent systems with MPA. In *Proceedings of the Second Workshop on Process Algebras and Performance Modelling* (PAPM), Erlangen, Germany, pages 175–189, 1994.
4. A. Giacalone, C.C Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of the Working Conference on Programming Concepts and Methods (IFIP TC2),* Sea of Galilee, Israel, 1990.
5. R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1996.
6. H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In *Proceedings of the* 10^{th} *IEEE Symposium on Real-Time Systems*, 1989.
7. C.C. Jou and S.A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In J.*Proceedings of CONCUR 90*, LNCS 458, pages 367–383, 1990.
8. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. In *Conference Record of the* 16^{th} *ACM Symposium on Principles of Programming Languages*, pages 344–352, 1989.
9. R. Milner. *Communication and Concurrency.* Prentice-Hall International, 1989.
10. A. Philippou, I. Lee, and O. Sokolsky. Weak bisimulation for probabilistic systems. In *Proceedings of CONCUR 2000*, LNCS 1877, pages 334–349, 2000.
11. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* Technical report MIT/LCS/TR-676. PhD thesis, MIT, Dept. of EECS, 1995.
12. R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. In *Proceedings of CONCUR 94*, LNCS 836, pages 481–496, 1994.
13. K. Seidel. Probabilistic communicating processes. Technical Report PRG-102, Ph.D. Thesis, Programming Research Group, Oxford University Computing Laboratory, 1992.
14. E.W. Stark and S.A. Smolka. A complete axiom system for finite-state probabilistic processes. In *Proof, Language and Interaction: Essays in Honour of Robin Milner.* MIT Press, 1999.
15. M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of 26th IEEE Symposium on Foundations of Computer Science*, pages 327–338, 1985.
16. W. Yi and K.G. Larsen. Testing probabilistic and nondeterministic processes. In *Protocol Specification, Testing and Verification XII*, pages 47–61, 1992.

Noninterference for Concurrent Programs*

Gérard Boudol and Ilaria Castellani

INRIA, 2004 Route des Lucioles, BP 93, 06902 Sophia-Antipolis, France.

Abstract. We propose a type system to ensure the property of *noninterference* in a system of concurrent programs, described in a standard imperative language extended with parallelism. Our proposal is in the line of some recent work by Irvine, Volpano and Smith. Our type system, as well as our semantics for concurrent programs, seem more natural and less restrictive than those originally presented by these authors. Moreover, we show how to adapt the type system in order to preserve the noninterference results in the presence of scheduling policies, while remaining in a nonprobabilistic setting.

1 Introduction

The aim of this paper is to study the notion of secure information flow, and more specifically of *noninterference* (a notion first introduced by Goguen and Meseguer in [4]) in the setting of concurrency. Our starting point is the paper [15] by Volpano, Smith and Irvine, and the subsequent paper [12] by Smith and Volpano, where noninterference is enforced by means of a simple type system in an imperative language with security levels. The language considered in [15] is purely sequential, and is extended in [12] with asynchronous parallelism (interleaving). In this introduction, and in the examples given in the paper, the security levels will simply be high and low. High-level variables are supposed to contain *secret* information, while low-level variables contain *public* information. However all results will be given for an arbitrary lattice of security levels.

In Volpano et al.'s work, *noninterference* means that variables of a given level do not interfere with those of lower levels: more precisely, the values of low-level variables are not dependent on the values of high-level variables. Noninterference is meant to model the absence of information flow from high level to low level. Such information flow is considered *insecure*, as it amounts to the disclosure of secret information into the public domain. Insecure flow can be *explicit*, when assigning the value of a high variable to a low variable, or *implicit*, when testing the value of a high variable and then assigning to a low variable, for instance. In the approach of [15,12], these situations are prevented by means of a type system. More precisely, explicit flow is prevented by requiring that the level of the assigned variable be at least as high as that of the source variable, while implicit flow is prevented by asking that the level of the commands in the branches of a conditional (the level of a command being that of its lowest assigned variables)

* Research partially funded by the EU Working Group CONFER II and by the french RNRT Project MARVEL.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 382–395, 2001.

$$\gamma : \texttt{if } PIN = 0 \texttt{ then } t_\beta := tt \texttt{ else } t_\alpha := tt$$
$$\alpha : \texttt{while } t_\alpha \neq tt \texttt{ do nil} ; \; r := 0 ; \; t_\beta := tt$$
$$\beta : \texttt{while } t_\beta \neq tt \texttt{ do nil} ; \; r := 1 ; \; t_\alpha := tt$$

PIN, t_α, t_β : boolean variables of type H

r : boolean variable of type L

γ : thread of type H, α, β : threads of type L

Fig. 1. Information Flow through Control Flow

be at least as high as that of the tested variable. Implicit flow can also arise in while-loops, and is prevented by a similar condition on the type of the body of the loop.

In fact, because of while-loops, the definition of noninterference is more precise than what is stated above: it says that no change in the values of low-level variables should be observed as a consequence of a change in high-level variables, *provided that the program terminates successfully.* Using subscripts to explicitly indicate the security level of a variable, consider the following program, that terminates if $x_H \neq 0$ and loops forever (doing nothing) otherwise:

$$\texttt{while } x_H = 0 \texttt{ do nil} ; \; y_L := 1 \tag{1}$$

Should this program be accepted, that is, should it be typable? According to the above definition of noninterference the answer is "yes", since whenever the program terminates it produces the same value $y_L = 1$ for its low-level variable. Indeed, this program is typable in Volpano and Smith's type system, since the loop is typable and the sequential composition of typable programs is always typable.

However, accepting such a program leads to problems when parallelism is introduced in the language. These problems can be concisely described as "disguising information flow as control flow". Let us illustrate the problem by means of an example, which is a simplified version of the *PIN* example given by Smith and Volpano in [12]. In this example, given in Figure 1, three threads α, β and γ are run (asynchronously) in parallel. There are four variables, a high-level variable *PIN* tested by thread γ, two high-level variables t_α and t_β serving as "triggers" for threads α and β, and a low-level variable r written by α and β. As can be easily seen, with initial values $t_\alpha = t_\beta = ff$ the effect of the program is to copy the value of the secret variable *PIN* into the public variable r. The illicit information flow from *PIN* to r is implemented through the *control flow* from γ to α or β. However, if we assume that a system of concurrent threads is typable provided each component is typable, this particular system is to be accepted.

To circumvent this problem, Smith and Volpano propose in [12] to forbid the use of high-level variables as guards in while-loops, that is, assuming that there is a lowest security level, to accept only while-loops of low level. While ruling

out the program in (1), and also the threads α and β of the *PIN* example, this solution seems a bit drastic. It excludes inoffensive programs such as `while` $x_H = 0$ `do nil`. We shall propose here a different solution to the problem raised by while-loops in the presence of parallelism, which allows this program to be typed, while ruling out the programs of example (1) and Figure 1. Our solution is based on the observation that a program such as

$$\texttt{while } x_H = 0 \texttt{ do nil}$$

should indeed be considered with some care in a concurrent setting, but only as a "guard", that is, as regards what may follow it. In the context of concurrent threads, if the control comes back to this `while` loop, this may be with a value for x_H different from 0, contrarily to what happens in a sequential setting. In other words, this program may observe the behaviour of other, concurrent components, in the course of their execution, and influence accordingly the behaviour of the thread in which it participates. Technically, this means that we will abandon the *big-step semantics* which is the basis of Volpano et al.'s analysis in favor of a *small step semantics* for programs, which is the approach usually adopted in dealing with parallelism. Our aim is then to ensure a stronger form of noninterference, where the course of values – not just the final value – of a low-level variable does not depend upon the value of high variables. Typically, the program (1) is no longer interference-free in this stronger sense. In order to reject it, we introduce a refinement of the type system, where the level of a guard – the expression tested by a `while` loop – is taken into account in sequential composition.

We will also examine the situation where a scheduling policy is in force in a thread system: we will introduce a few new programming primitives to describe formally such a situation, and show how to adapt the type system for this new setting, where new interference phenomena arise. As can be expected, this will result in a slight restriction on the type of certain programs, though not as severe as that prefigured in [12].

The rest of the paper is organised as follows. In Section 2 we introduce the language, its operational semantics and its type system. Section 3 presents the properties of typed programs, including subject reduction and noninterference. Finally, in Section 4 we consider the extended language with scheduling policies. The proofs are omitted from this extended abstract. They are to be found in the full version of the paper [2].

2 The Language and Type System

The language we consider is essentially that of [12] (where e stands for a boolean or arithmetic expression, whose syntax we do not detail here). We use the following two-level syntax, where U, V denote sequential programs, while P, Q denote general (concurrent) programs:

$$U,\ V \ldots := \texttt{nil} \mid x := e \mid U; V \mid \texttt{if } e \texttt{ then } U \texttt{ else } V \mid \texttt{while } e \texttt{ do } U$$

$$P,\ Q \ldots := U \mid U; P \mid \texttt{if } e \texttt{ then } P \texttt{ else } Q \mid \texttt{while } e \texttt{ do } P \mid P \parallel Q$$

Note that on the left of a sequential composition, we must have a sequential program. Thus programs of the form $(P \parallel Q); R$ are not allowed. With this restriction, our language is still more general than that of [12], which describes concurrent systems as collections of threads, thus allowing only top-level parallelism, while we allow the dynamic spawning of new threads.

The operational semantics of the language is given in terms of transitions between configurations $(P, \mu) \rightarrow (P', \mu')$ where P, P' are programs and μ, μ' stand for *memories*, that is mappings from variables to values. These mappings are extended in the obvious way to expressions, whose evaluation is assumed to be atomic as in [12]. We use the notation $\mu[v/x]$ for memory update. The rules specifying the operational semantics of programs are presented in Figure 2. As pointed out already in the introduction, the semantics used here is a *small step semantics*, as opposed to the *big step semantics* of [12][1]. The rules are fairly standard, and we shall not comment on them.

In the introduction we argued that, in a small-steps semantics, the program (1) should be treated as another case of implicit information flow. Intuitively, when exiting a loop one gets some information about its guard; it seems then appropriate to require that what follows the loop – its "continuation" – have level at least as high as that of the loop guard. This will be the basic idea of our new type system, which is closely inspired by that given by Volpano et al. in [15] – however as suggested by the above example it will be more restrictive than that of [15] on the sequential sublanguage, because of our more detailed observation of programs.

The types of data and expressions are *security levels*, that is elements of a lattice $(\mathcal{S}, \leq)$. We denote the operations of meet and join respectively by $\sqcap$ and $\sqcup$. These types are ranged over by τ, σ. In the examples, the lattice of security levels will simply be $\{L, H\}$, with $L < H$. The types of variables (when used in the left-hand side of an assignment) are of the form $\tau\ var$. Our first point of departure from [15] concerns the types for programs. Type judgements in [15] are of the form $\Gamma \vdash P : \tau\ cmd$, where Γ is a mapping from variables to types of variables, i.e. elements of $\{\tau\ var \mid \tau \in \mathcal{S}\}$. The meaning of $\Gamma \vdash P : \tau\ cmd$ is that in the type environment Γ, τ is a *lower bound* for the level of the *assigned variables* of P. In line with this intuition, subtyping for programs is contravariant, that is $\tau\ cmd \leq \tau'\ cmd$ if $\tau' \leq \tau$. Thus for instance any program of type $H\ cmd$ can be downgraded to type $L\ cmd$. A program of type $H\ cmd$ is guaranteed not to contain any assignment to a low variable.

To take into account loop guards, we shall use here more refined types $(\tau, \sigma)\, cmd$, where the first component τ plays the same rôle as in the type $\tau\, cmd$, while the second component σ is the *guard type*, an *upper bound* on the level of the loop guards occurring in a program. Accordingly, the subtyping for programs is contravariant in its first component and covariant in the second:

$$(\tau, \sigma)\ cmd \leq (\tau', \sigma')\ cmd \quad \text{if} \quad \tau' \leq \tau \ \text{ and } \ \sigma \leq \sigma'$$

[1] In fact, the semantics of [12] is a mixture of small and big step semantics: transitions are given between configurations but there are two kinds of configurations, intermediate and final ones, suggesting that termination should be observed.

$$\text{(Assign-Op)}\quad \frac{}{(x := e, \mu) \to (\mathtt{nil}, \mu[\mu(e)/x])}$$

$$\text{(Seq-Op1)}\quad \frac{(U, \mu) \to (U', \mu')}{(U; P, \mu) \to (U'; P, \mu')}$$

$$\text{(Seq-Op2)}\quad \frac{(P, \mu) \to (P', \mu')}{(\mathtt{nil}; P, \mu) \to (P', \mu')}$$

$$\text{(Cond-Op1)}\quad \frac{\mu(e) = tt}{(\mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q, \mu) \to (P, \mu)}$$

$$\text{(Cond-Op2)}\quad \frac{\mu(e) \neq tt}{(\mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q, \mu) \to (Q, \mu)}$$

$$\text{(While-Op1)}\quad \frac{\mu(e) = tt}{(\mathtt{while}\ e\ \mathtt{do}\ P, \mu) \to (P; \mathtt{while}\ e\ \mathtt{do}\ P, \mu)}$$

$$\text{(While-Op2)}\quad \frac{\mu(e) \neq tt}{(\mathtt{while}\ e\ \mathtt{do}\ P, \mu) \to (\mathtt{nil}, \mu)}$$

$$\text{(Par-Op1)}\quad \frac{(P, \mu) \to (P', \mu')}{(P \parallel Q, \mu) \to (P' \parallel Q, \mu')}$$

$$\text{(Par-Op2)}\quad \frac{(Q, \mu) \to (Q', \mu')}{(P \parallel Q, \mu) \to (P \parallel Q', \mu')}$$

Fig. 2. Operational Semantics for Parallel Programs

The guard type will be set up by while-loops and looked up by sequential composition. The complete type system for programs is shown in Figure 3. Notice that the guard type plays no particular rôle in rules (Nil), (Assign) and (Cond), which are plain adaptations of the ones in [15]. Let us comment a little on the rules for while-loops and sequential composition, which are the main novelty w.r.t. [15,12]. As explained, the guard type is σ for a while-loop testing an expression of level σ, and from then onwards it should stay equal to σ to prevent concatenation with low-level programs. Rule (Seq) is precisely designed to avoid sequencing "low" assignments after a program with "high" guards. This rules out the kind of implicit flow exhibited by the program (1). One may notice that rule (While) imposes types of the form $(\tau, \tau)\, cmd$ to while-loops (by subtyping they also have types $(\theta, \sigma)\, cmd$ with $\theta \leq \sigma$). We let the reader check that, had we accepted for instance $(H, L)\, cmd$, we would not avoid interferences, as shown by the example

$$\text{(Nil)}\quad \frac{}{\Gamma \vdash \texttt{nil} : (\tau, \sigma)\ cmd}$$

$$\text{(Assign)}\quad \frac{\Gamma \vdash e : \tau, \quad \Gamma(x) = \tau\ var}{\Gamma \vdash x := e : (\tau, \sigma)\ cmd}$$

$$\text{(Seq)}\quad \frac{\Gamma \vdash U : (\tau, \sigma)\ cmd, \quad \Gamma \vdash P : (\tau', \sigma')\ cmd, \quad \sigma \leq \tau'}{\Gamma \vdash U\,;\, P : (\tau \sqcap \tau', \sigma \sqcup \sigma')\ cmd}$$

$$\text{(Cond)}\quad \frac{\Gamma \vdash e : \tau, \quad \Gamma \vdash P : (\tau, \sigma)\ cmd, \quad \Gamma \vdash Q : (\tau, \sigma)\ cmd}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q : (\tau, \sigma)\ cmd}$$

$$\text{(While)}\quad \frac{\Gamma \vdash e : \tau, \quad \Gamma \vdash P : (\tau, \tau)\ cmd}{\Gamma \vdash \texttt{while}\ e\ \texttt{do}\ P : (\tau, \tau)\ cmd}$$

$$\text{(Par)}\quad \frac{\Gamma \vdash P : (\tau, \sigma)\ cmd, \quad \Gamma \vdash Q : (\tau, \sigma)\ cmd}{\Gamma \vdash P \parallel Q : (\tau, \sigma)\ cmd}$$

$$\text{(Subtyping)}\quad \frac{\Gamma \vdash P : (\tau, \sigma)\ cmd, \quad \tau' \leq \tau, \quad \sigma \leq \sigma'}{\Gamma \vdash P : (\tau', \sigma')\ cmd}$$

Fig. 3. Typing Rules for Concurrent Programs

$$\begin{array}{l} \texttt{if}\ \ x_H = 0 \quad \texttt{then while}\ \ y_L = 0\ \ \texttt{do nil} \\ \qquad\qquad\quad \texttt{else nil}\,; \\ u_L := u_L + 1 \end{array} \tag{2}$$

Similarly, we have to rule out the insecure program

$$\texttt{while}\ \ tt\ \ \texttt{do}\ \ (\, y_L := y_L + 1\ ;\ \texttt{while}\ \ x_H = 0\ \ \texttt{do nil}\,) \tag{3}$$

and this shows why loops having $(L, H)\ cmd$ as their unique type should be forbidden.

3 Properties of Typed Programs

In this section we prove some desired properties of our type system. The first property, *subject reduction*, states that types are preserved along execution.

Theorem 3.1. (Subject Reduction)
If $\Gamma \vdash P : (\tau, \sigma)\ cmd$ *and* $(P, \mu) \to (P', \mu')$, *then* $\Gamma \vdash P' : (\tau, \sigma)\ cmd$.

Proof: By induction on the inference of $\Gamma \vdash P : (\tau, \sigma)\ cmd$, and then case analysis on the last rule used in this inference. □

We shall use the following assumptions about expressions:

Assumption 3.2 (Termination of Expression Evaluation)
For any memory μ and expression e, the value $\mu(e)$ is defined.

Assumption 3.3 (Simple Security)
If $\Gamma \vdash e : \tau$, then every variable occurring in e has type $\tau'\,var$ in Γ, with $\tau' \leq \tau$.

We introduce now, for any type environment Γ and security level ω, a notion of equality on memories which formalises the idea that two memories coincide on variables of level less than or equal to ω in Γ. Intuitively, such memories are indistinguishable for an observer of level ω.

Definition 3.4 (ω-Equality of Memories)
$\mu =^{\omega}_{\Gamma} \nu \;\Leftrightarrow_{\rm def}\; \forall x.\ \Gamma(x) = \tau\, var \;\&\; \tau \leq \omega \;\Rightarrow\; \mu(x) = \nu(x)$.

Definition 3.5 ((Γ,ω)-Bisimulation)
A relation $\mathcal{R}$ on configurations is a (Γ,ω)-bisimulation if $(P,\mu)\,\mathcal{R}\,(Q,\nu)$ implies

(i) $\mu =^{\omega}_{\Gamma} \nu$
(ii) $(P,\mu) \to (P',\mu') \;\Rightarrow\; \exists Q',\nu'.\ (Q,\nu) \to^* (Q',\nu') \;\wedge\; (P',\mu')\,\mathcal{R}\,(Q',\nu')$
(iii) $(Q,\nu) \to (Q',\nu') \;\Rightarrow\; \exists P',\mu'.\ (P,\mu) \to^* (P',\mu') \;\wedge\; (P',\mu')\,\mathcal{R}\,(Q',\nu')$

The (Γ,ω)-*bisimulation equivalence* on configurations, noted $\approx^{\omega}_{\Gamma}$, is the largest (Γ,ω)-bisimulation.

The following two lemmas confirm the intuition, discussed earlier, behind the type judgements $\Gamma \vdash P : (\tau,\sigma)\,cmd$.

Lemma 3.6 (Confinement)
If $\Gamma \vdash P : (\tau,\sigma)\,cmd$ then every variable assigned to in P has type $\theta\,var$ in Γ, with $\tau \leq \theta$.

Lemma 3.7 (Guard Safety)
If $\Gamma \vdash P : (\tau,\sigma)\,cmd$ then every loop guard in P has type θ in Γ, with $\theta \leq \sigma$.

Definition 3.8 (ω-Boundedness)
A program P is ω-*bounded* if $\Gamma \vdash P : (\tau,\sigma)\,cmd$ implies $\tau \leq \omega$.

Definition 3.9 (ω-Guardedness)
A program P is ω-*guarded* if there exist τ,σ, with $\sigma \leq \omega$, such that $\Gamma \vdash P : (\tau,\sigma)\,cmd$.

Note that by the Confinement Lemma, a program which is *not* ω-*bounded* cannot write on variables of level less than or equal to ω. Similarly, by the Guard Safety Lemma, a program which is ω-*guarded* does not contain loop guards of level higher than or incomparable with ω. As a consequence of subject reduction, both *non*-ω-boundedness and ω-guardedness are preserved by execution.

Proposition 3.10 (Bisimilarity of Non ω-Bounded Programs)
Let $\mathcal{S}^{\Gamma,\omega}$ be the relation consisting of the pairs $((P,\mu),(Q,\nu))$ such that $\mu =^{\omega}_{\Gamma} \nu$ and there exist τ, σ and τ', σ' with $\tau \not\leq \omega$, $\tau' \not\leq \omega$, such that $\Gamma \vdash P : (\tau,\sigma)\,cmd$ and $\Gamma \vdash Q : (\tau',\sigma')\,cmd$. Then $\mathcal{S}^{\Gamma,\omega}$ is a (Γ,ω)-bisimulation.

Proof: Let $((P,\mu),(Q,\nu)) \in \mathcal{S}^{\Gamma,\omega}$ and $(P,\mu) \to (P',\mu')$. This can be matched by $(Q,\nu) \to^{*} (Q,\nu)$, since by the Confinement Lemma $\mu' =^{\omega}_{\Gamma} \mu =^{\omega}_{\Gamma} \nu$, and by the Subject Reduction Theorem $\Gamma \vdash P' : (\tau,\sigma)$. □

Note that (Γ,ω)-bisimilarity does not preserve termination. For instance, for any memories μ and ν such that $\mu =^{\omega}_{\Gamma} \nu$ we have:

$$(\,\texttt{nil},\,\mu\,) \;\approx^{\omega}_{\Gamma}\; (\,\texttt{while}\ tt\ \texttt{do nil},\,\nu\,)$$

We introduce now a notion which will play a key rôle for noninterference.

Definition 3.11 (ω-Constrainment)
A program P is *ω-constrained* if there exist τ,σ, with $\tau \not\leq \omega$ and $\sigma \leq \omega$, such that $\Gamma \vdash P : (\tau,\sigma)\,cmd$.

By definition any ω-constrained program is both ω-guarded and *not* ω-bounded. It is worth stressing that the converse is not true, as shown by the program `while` tt `do nil`. Clearly, for any type environment Γ, this program is ω-guarded for any security level ω and not ω-bounded if $\omega \neq \top$. However it is not ω-constrained, as a consequence of the uniform typing in rule (While). Indeed, an important property of ω-constrained programs is the following.

Lemma 3.12 (Termination of ω-Constrained Sequential Programs)
If U is ω-constrained, then for any μ there exist μ', U' such that $(U,\mu) \to^{*} (U',\mu')$ and $U' = \texttt{nil};\cdots;\texttt{nil}$.

Finally we can state our main result:

Theorem 3.13. (Noninterference)
If P is typable in Γ, then $(P,\mu) \approx^{\omega}_{\Gamma} (P,\nu)$ for any μ,ν such that $\mu =^{\omega}_{\Gamma} \nu$.

Proof: We define inductively the relation $\mathcal{R}_0^{\Gamma,\omega}$ on configurations as follows: $(P,\mu)\,\mathcal{R}_0^{\Gamma,\omega}\,(Q,\nu)$ if and only if P and Q are typable, $\mu =^{\omega}_{\Gamma} \nu$ and one of the following holds:

1. $(P,\mu) \approx^{\omega}_{\Gamma} (Q,\nu)$
2. $P = Q$ and P is ω-bounded
3. $P = U;R$ and $Q = V;R$, where both U and V are ω-constrained
4. $P = U;R$ and $Q = V;R$, where $(U,\mu)\,\mathcal{R}_0^{\Gamma,\omega}\,(V,\nu)$ and R is not ω-bounded
5. P is not ω-bounded and $Q = V;R$, where $(\texttt{nil},\mu)\,\mathcal{R}_0^{\Gamma,\omega}\,(V,\nu)$ and R is not ω-bounded (or symmetrically)
6. $P = P_1 \parallel P_2$ and $Q = Q_1 \parallel Q_2$ with $(P_i,\mu)\,\mathcal{R}_0^{\Gamma,\omega}\,(Q_i,\nu)$.

We show that $\mathcal{R}_0^{\Gamma,\omega}$ is a (Γ,ω)-bisimulation. The theorem will be a consequence of this fact, since if P is typable, then either P is not ω-bounded, in which case $(P,\mu) \approx^{\omega}_{\Gamma} (P,\nu)$ by Proposition 3.10, or P is ω-bounded and $(P,\mu)\,\mathcal{R}_0^{\Gamma,\omega}\,(P,\nu)$ by the second clause of the definition. In the case of Clause 3, we use the Lemma 3.12. □

4 Adding a Scheduler

As pointed out by Smith and Volpano in [12], noninterference results such as those of the previous section rely on the hypothesis of a purely nondeterministic execution of concurrent programs. These results would break down if particular scheduling policies were enforced. We recall the example given in [12]. Assume a *round robin time slicing* scheduler, with a time slice of t steps, $t \geq 2$, and consider the composition $P = \alpha \parallel \beta$ of the following two threads:

$$\begin{aligned} \alpha &: \texttt{if}\ \ x_H = 0\ \texttt{then}\ Q\ \texttt{else nil}; \\ & \quad y_L := 0 \\ \beta &: y_L := 1 \end{aligned} \tag{4}$$

Then, supposing that Q is a convergent program that takes at least $t-1$ steps to execute, and that the scheduler gives precedence to α, the value of y_L will depend on that of x_H. The solution proposed in [12] to preserve noninterference in the presence of an arbitrary scheduler consists in forbidding conditionals with high guards[2], that is, again assuming that there is a lowest security level, to accept only conditional branching on low level expressions. This condition, combined with the exclusion of loops with high guards, required for multi-threading, resulted in [12] in a very severe limitation: the impossibility for any program to test a variable, except at the lowest level.

We present here a different solution for scheduling, which does not rule out conditionals with high guards. To this end we first formalise what it means for a system of concurrent programs to be controlled by a scheduler. Essentially, this means running the system in lockstep with a program that implements the scheduling policy. To describe controlled execution, we use a construction $P[Q]$, which makes P and Q move hand in hand, but allows the controller, P, to proceed by itself whenever Q is unable to move. Then a system consisting of n parallel programs P_i controlled by a scheduler *Sched* will be described as:

$$Sched\ [P'_1 \parallel \cdots \parallel P'_n]$$

where the P'_i are adaptations of the P_i, so that the processes can be triggered and suspended by the scheduler. To this end we introduce a new construct $\texttt{when}\ e\ \texttt{do}\ P$, whose semantics is that P is allowed to proceed, for one step, when the condition e holds. It is technically convenient to introduce another level in the syntax: besides the programs P, written according to the grammar given in Section 2, there is a set of "systems" S, T built as follows:

$$S\ ::=\ P\ \mid\ S \parallel T\ \mid\ S[T]\ \mid\ \texttt{when}\ e\ \texttt{do}\ S$$

Letting $w(S)$ denote the set of variables written (assigned to) by S, the construct $S[T]$ is only legal under the condition $w(S) \cap w(T) = \emptyset$.

[2] It is also suggested there that a better approach to scheduling would be *probabilistic*. Indeed a whole line of research on probabilistic noninterference has been developed, but this will not be our concern here, where we stick to a *possibilistic* setting.

$$\text{(Control-Op1)}\quad \frac{(S,\mu)\to(S',\mu'),\ (T,\mu)\to(T',\mu'')}{(S[T],\ \mu)\to(S'[T'],\mu'\sqcup_\mu\mu'')}$$

$$\text{(Control-Op2)}\quad \frac{(S,\mu)\to(S',\mu'),\quad (T,\mu)\not\to}{(S[T],\mu)\to(S'[T],\mu')}$$

$$\text{(When-Op)}\quad \frac{\mu(e)=tt,\qquad (S,\mu)\to(S',\mu')}{(\texttt{when}\ e\ \texttt{do}\ S,\ \mu)\to(\texttt{when}\ e\ \texttt{do}\ S',\ \mu')}$$

Fig. 4. Additional Operational Rules for Systems

$$\text{(Control)}\quad \frac{\Gamma\vdash S:(\tau,\sigma)\,cmd,\quad \Gamma\vdash T:(\tau,\sigma)\,cmd,\quad \tau\geq\sigma}{\Gamma\vdash S[T]\ :\ (\tau,\sigma)\,cmd}$$

$$\text{(When)}\quad \frac{\Gamma\vdash e:\theta,\quad \Gamma\vdash S:(\tau,\sigma)\,cmd,\quad \theta\leq\tau}{\Gamma\vdash \texttt{when}\ e\ \texttt{do}\ S:(\tau,\theta\sqcup\sigma)\,cmd}$$

Fig. 5. Additional Typing Rules for Systems

Notation 4.1 *We use* $(S,\mu)\to$ *to mean* $\exists S',\mu'$ *such that* $(S,\mu)\to(S',\mu')$*, and* $(S,\mu)\not\to$ *for the negation of* $(S,\mu)\to$*.*

The semantics of the new constructs is given in Figure 4, where $\mu'\sqcup_\mu\mu''$ represents the memory μ with the conjunction of the updates operated by S and by T, that is $\mu'\backslash\mu\cup\mu''\backslash\mu\cup(\mu'\cap\mu'')$. Then for instance the scheduled programs may be written $P_i'=\texttt{when}\ s_i\ \texttt{do}\ P_i$ where s_i is the "*proceed*" signal for program P_i, set up by the scheduler. The following program

$$Sched\ {}^{t}_{n} = i:=0\,;\ \texttt{while}\ tt\ \texttt{do}\ i\ :=\ [i+1]_{\mathrm{mod}\ n}; k\ :=\ 0;$$
$$\texttt{while}\ k<t\ \texttt{do}\ s_i:=tt; s_i:=ff; k:=k+1$$

describes a scheduler for a system of n threads, implementing round robin with time slice t, provided that all the s_i's are initially false. It is easy to imagine how to program other scheduling policies in a similar style.

The typing rules for the new operators are given in Figure 5. The side-conditions in rules (Control) and (When) need some comments. First, note that a `when` statement can induce an implicit flow, just like the conditional and `while` statements, as for instance in the system:

$$\texttt{when}\ x_H=0\ \texttt{do}\ y_L:=y_L+1$$

This explains the requirement $\theta \leq \tau$ in rule (When). On the other hand, the condition that the guard of the `when` statement should affect its guard type may seem superfluous at first sight, since a `when` statement can never be followed (in sequential composition) by any other system. The reason for this condition is that in a controlled system $S[T]$, a *blocked* behaviour of the controller S can create interferences if the controlled system T is low, and this blocked behaviour of S may be due to a `when` statement. Consider for instance the system $S[T]$ where P is a high program that does at least one step:

$$S = \texttt{when}\ \ x_H = 0\ \ \texttt{do}\ \ P$$
$$T = y_L := y_L + 1$$

We let the reader check that this system can lead to interference. Now if the `when` statement S were allowed to have type $(L, L)\, cmd$, the whole system $S[T]$ would be typable.

As regards the rule (Control), the condition $\tau \geq \sigma$ excludes for instance – if the security levels are L and H – systems $S[T]$ whose unique type is $(L, H)\, cmd$. Consider for instance the controlled system $S'[T']$, where:

$$S' = \texttt{while}\ x_H = 0\ \texttt{do nil}$$
$$T' = y_L := 0\ ;\ y_L := y_L + 1$$

Here again there is a possible interference due to the blocking of the controller after one step if the guard of the loop is false. Note that the only possible type of $S'[T']$ would be indeed $(L, H)\, cmd$, since it affects a low variable and has a high loop guard.

To extend our noninterference result to the new setting, we also need to restrict the typing rule for conditional branching, recording the tested expression as a guard (note the similarity with the rule for the `when` statement):

(Cond-Strict)

$$\frac{\Gamma \vdash e : \theta, \quad \Gamma \vdash P : (\tau, \sigma)\, cmd, \quad \Gamma \vdash Q : (\tau, \sigma)\, cmd, \quad \theta \leq \tau}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q : (\tau, \theta \sqcup \sigma)\, cmd}$$

This rules out for instance the thread α of our initial example (4), because a low assignment can no longer be performed after a high test.

It is easy to check that the Subject Reduction Theorem and the Confinement Lemma extend to the new language. Similarly, the definitions of (Γ, ω)-bisimulation and ω-boundedness remain formally the same as those for the base language (modulo the replacement of programs by systems). Obviously, the Guard Safety Lemma may now be strengthened into:

Lemma 4.2 (Strong Guard Safety)
If $\Gamma \vdash S : (\tau, \sigma)\, cmd$ then every loop, conditional or when statement guard in S has type θ in Γ, with $\theta \leq \sigma$.

Lemma 4.3 (Deterministic Behaviour of ω-Guarded Systems)
If S is ω-guarded in Γ and $\mu =^{\omega}_{\Gamma} \nu$, then $(S,\mu) \to (S',\mu')$ implies $(S,\nu) \to (S',\nu')$, with $\mu' =^{\omega}_{\Gamma} \nu'$.

We are now able to generalise our noninterference result.

Theorem 4.4. (Extended noninterference)
If S is typable in Γ, then $(S,\mu) \approx_{\Gamma} (S,\nu)$ for any μ,ν such that $\mu =_{\Gamma} \nu$.

Proof: We define inductively the relation $\mathcal{R}_1^{\Gamma,\omega}$ as follows: $(S,\mu)\,\mathcal{R}_1^{\Gamma,\omega}\,(T,\nu)$ if and only if S and T are typable, $\mu =^{\omega}_{\Gamma} \nu$ and one of the following holds:

1. $(S,\mu)\,\mathcal{R}_0^{\Gamma,\omega}\,(T,\nu)$, where $\mathcal{R}_0^{\Gamma,\omega}$ is the relation considered in the proof of Theorem 3.13
2. $(S,\mu) \approx^{\omega}_{\Gamma} (T,\nu)$
3. $S = T$ and S is ω-bounded
4. $S = S_0 \parallel S_1$, $T = T_0 \parallel T_1$ and $(S_i,\mu)\mathcal{R}_1^{\Gamma,\omega}\,(T_i,\nu)$
5. $S = \texttt{when}\ e\ \texttt{do}\ S_1$, $T = \texttt{when}\ e\ \texttt{do}\ T_1$, $\Gamma(e) \leq \omega$ and $(S_1,\mu)\mathcal{R}_1^{\Gamma,\omega}\,(T_1,\nu)$

Then we show that $\mathcal{R}_1^{\Gamma,\omega}$ is a (Γ,ω)-bisimulation. In Clause 3, for the case of the control construct, we use the Lemma 4.3. □

5 Conclusion and Related Work

We have addressed the question of secure information flow in systems of concurrent programs. This covers one of the security problems that can arise, for instance, when a mobile program visits different sites, namely that of preserving the *confidentiality* of the visited sites' private data. In fact, in [3], it is shown how a form of noninterference called *non deducibility on composition* may be used to model also other security properties like *authenticity*, *non repudiation* and *fairness*. Noninterference thus appears as a rather interesting notion to study when security is concerned. On the other hand, it may be argued [8] that *covert channels*, that is implicit information flows of the kind considered here, are unavoidable in practice, as they can arise also at the hardware level. Thus the aim of statically ensuring the absence of covert channels might be a hard one to realise. We certainly do not claim here to cover the whole range of possible attacks from a hostile party.

The issue of noninterference has been largely studied in the literature, using different models, and it is not our intention here to review the various approaches. We focussed on the approach of Volpano et al., as it applies to a fairly standard language, which can be assumed to be the kernel of more sophisticated practical languages.

The question of secure flow and noninterference has also started to be investigated in the setting of process calculi, and in particular in mobile process calculi [6], [7], [10] and [5]. The treatment in the first two papers seems however overly restrictive: it amounts (at least in the core calculus) to forbid all control flow from actions on high channels to actions on low channels. In [7], the

core calculus is extended with more sophisticated constructs; in the extended calculus some actions may be classified as "innocuous", and the restriction on control flow may be relaxed when these actions are involved. The last two papers, [10] and [5], are less restrictive and closer in spirit to our approach, as they try to distinguish the dangerous control flow (implementing information flow) from the harmless control flow which should not be restricted. Another related paper is [1], which studies secrecy properties in security protocols expressed in the *spi*-calculus.

As concerns noninterference in the presence of scheduling policies, the most popular approach has been so far the probabilistic one, taken for instance in [14] and [11]. Our stand here was to handle scheduling within a possibilistic setting.

An issue which has not been addressed here, but is planned for future work, is the feasibility of checking noninterference using a type inference algorithm, in the line of [13]. Current work is also oriented towards the treatment of more realistic languages, as advocated for instance in [9], including exceptions and some form of higher-order.

Acknowledgements. We would like to thank the anonymous referees for helpful comments.

References

[1] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.
[2] Gérard Boudol and Ilaria Castellani. Noninterference for concurrent programs. Research report, INRIA, 2001.
[3] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proceedings ICALP'00*, number 1853 in LNCS, 2000.
[4] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings 1982 IEEE Symposium on Security and Privacy*, pages 11–20, 1982.
[5] M. Hennessy. The security π-calculus and noninterference. Computer Science Technical Report 2000:05, University of Sussex, 2000.
[6] M. Hennessy and J. Riely. Information flow vs resource access in the asynchronous pi-calculus (extended abstract). In *Proceedings ICALP'00*, number 1853 in LNCS, 2000.
[7] K. Honda, V. Vasconcelos, and N. Yoshida. Secure information flow as typed process behaviour. In *Proceedings ESOP'00*, number 1782 in LNCS, 2000.
[8] J. Millen. 20 years of covert channel modeling and analysis. In *IEEE Symposium on Security and Privacy*, 1999.
[9] A. Myers. Jflow: Practical mostly-static information flow control. In *26th ACM Symposium on Principles of Programming Languages (POPL)*, 1999.
[10] F. Pottier and S. Conchon. Information flow inference for free. In *Proceedings ICFP'00*, 2000.
[11] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *IEEE Computer Security Foundations Workshop*, 2000.
[12] G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In ACM, editor, *Proceedings POPL '98*, pages 355–364. ACM Press, 1998.

[13] D. Volpano and G. Smith. A type-based approach to program security. In *TAPSOFT'97*, number 1214 in LNCS, pages 607–621, 1997.

[14] D. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. *Journal of Computer Security*, 7(2-3), 1999.

[15] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

Distributed Controller Synthesis for Local Specifications

P. Madhusudan[1] and P.S. Thiagarajan[2]

[1] The Institute of Mathematical Sciences madhu@imsc.ernet.in
[2] CREST, School of ECE, Georgia Institute of Technology, Atlanta, USA (On leave from Chennai Mathematical Institute, Chennai, India) thiagu@ece.gatech.edu

Abstract. We consider the problem of synthesizing distributed controllers for reactive systems against local specifications. We show that a larger class of architectures become decidable in comparison to the analogous problem for global specifications. We identify the exact class of architectures for which the problem is decidable. Our results also show the decidability of a related realizability problem for local specifications.

1 Introduction

An *open* reactive system is one which interacts with its environment on a systematic basis and whose behaviour crucially depends on this interaction. The key feature of open systems is that one is required to distinguish between the capabilities of the system and its environment. Typically, one views such an open system as getting inputs from the environment and reacting to it by outputting values. *Realizability* and *controller synthesis* problems arise naturally in the study of open systems.

The realizability problem is to determine, given a specification of an open system, say as a temporal logic formula, whether there exists a program for the system such that no matter how the environment behaves, the overall behaviour satisfies the specification. The program will fix the value the system outputs on receiving a particular input and this choice can depend on the past history of the interaction with the environment. The environment is allowed to input any value at any point.

The controller synthesis problem on the other hand, starts with an open system — often called a *plant* in this context — and a specification, say, a temporal logic formula. The plant is viewed as an existing program which specifies the ways in which the system can react to its inputs. The goal now is to come with a strategy to interact with the environment in a way allowed by the plant, such that all behaviours satisfy the specification. Thus the strategy, in this setting, acts as a *controller* for the plant which restricts the behaviours of the plant so that the specification is met.

There is a wealth of literature on realizability and controller synthesis problems for open systems as evidenced in [BL69,Rab72,Tho95,RW89,PR89,ALW89] for linear-time specifications and [KV96,MT98,KV99,KMTV00] for branching-time specifications. All of these studies are confined to programs and plants that

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 396–407, 2001.

consist of a *single* sequential module. However, these problems often arise in a distributed context and here, the main results presently available are due to Pnueli and Rosner [PR90]. We extend here their results and the point of departure is to consider *local* specifications. Before going into this in more detail, we wish to mention the work reported in [dAHM00,dAH00,AHK97] which are also concerned with control-related issues in a distributed setting but not directly connected to the concerns of the present work.

Pnueli and Rosner consider distributed programs by using the notion of an *architecture* which basically consists of a set of sites and some communication channels between them. The sites also have external input and output channels through which they interact with the environment. The specification consists of linear time temporal logic formulas whose atomic propositions can state properties of the values on the external input and output channels. The specification is hence a *global* one which can talk about simultaneous channel-values at different sites. The surprising main result is that even in the case of an architecture consisting of *two* sites and *no* internal communication channels, the realizability problem is undecidable. They also consider pipeline architectures which consists of a linear array of sites $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n$ with an external input channel allowed only for s_0. They show that the realizability problem is decidable for this class of architectures.

It turns out that many results of [PR90] extend also to the control-synthesis problem that we wish to study. Here, we are given in addition to the specification, a set of programs at each site and the problem is to come up with *local* strategies for the sites to restrict the programs; by a local strategy at a site, we mean a strategy which only knows the values which the input channels to this site have carried so far. In [PR90], realizability is actually shown to be decidable for a larger class of architectures (called *hierarchical architectures*) but only the decidability results for pipelines carry over for the control-synthesis problem.

In this paper, we drop global specifications since they turn out to be unreasonably expressive, and instead consider local specifications and, for convenience, study only the controller synthesis problem. We identify a special class of architectures called *clean* pipelines (see Figure 1) which are just like the pipelines mentioned above except external inputs are also allowed for the right-end site s_n. Our main result is that the controller synthesis problem for local specifications is decidable for an architecture $\mathcal{A}$ iff each connected component of $\mathcal{A}$ is a clean pipeline or is a sub-architecture of a clean pipeline.

Our undecidability results go through for weaker acceptance conditions down to reachability. Thus our negative results show that even in the presence of local specifications, the controller synthesis problem is intractable for almost all architectures. However, on the positive side, our results show that for local specifications, one can handle the nontrivial distributed reactive system which consists of two sites, both of them interacting with the environment, and with an internal channel from one to the other. Indeed, our results show that the realizability problem can also be effectively solved in this important setting,

where the specifications at the sites *can state properties of the internal channels as well*.

The undecidability result of [PR90] follows from the study of multi-player games of incomplete information by Peterson and Reif [PR79]. In this context, our results show that there are certain games where two players, playing against an adversary, have incomplete information about each other, and yet determining whether they have a winning strategy is decidable.

In the next section we introduce the formal setting for our work, Section 3 establishes our decidability results and in Section 4 we prove the undecidability results. Due to lack of space, we provide only the main proof ideas — more details can be found in [MT01].

2 Problem Setting

An *architecture* is a tuple $\mathcal{A} = (S, X, T, r, w)$ where $S = \{s_1, \ldots, s_k\}$ is a finite nonempty set of *sites*, $X = \{x_1, \ldots, x_l\}$ is a set of external (or environment) input channels and $T = \{t_1, \ldots, t_n\}$ is a set of internal channels. r is a function $r : X \cup T \to S$ which identifies for each channel a process which *reads* the channel; $w : T \to S$ identifies for each internal channel, a process which writes into it. We assume, without loss of generality, that each process has at most one external input channel and that there is at most one channel from one site to another.

We represent architectures graphically as directed graphs whose nodes are the sites and every channel $z \in X \cup T$ is represented by an edge — if $z \in T$, then it is an edge from $w(z)$ to $r(z)$ and if $z \in X$, then it is a sourceless edge to $r(z)$. We only consider *acyclic* architectures — i.e. those architectures whose graph representation does not have a directed cycle. We assume further that every site has at least one input (external or internal) channel.

In our framework, each site runs a program which reads its external and internal channel inputs and reacts by sending outputs along the internal channels to other processes and changing its state. The moves are synchronous — i.e. the programs read a set of external inputs and make one collective move while respecting the partial order imposed by the architecture.

For example, in the architecture $\mathcal{A}_2$ in Figure 2, in a synchronous step, s_1 will read the environment's input on x_1, change its state and write onto t_1; s_2 will read this input on t_1 and the input on x_2, change its state and write onto t_2; s_3 will read this value and change its internal state.

For a site $s \in S$, let $in(s) = r^{-1}(s)$, the set of channels which s reads from and $out(s) = w^{-1}(s)$, the set of channels S writes into. Given an architecture $\mathcal{A}$, a *domain definition* for $\mathcal{A}$ is a function D which associates with each $z \in X \cup T$ a finite set of values which can be sent along the channel z. We denote $D(z)$ sometimes as D_z. For a set of channels Z, a *valuation function for* Z is a function h whose domain is Z and which maps each $z \in Z$ to an element of D_z. Let $\mathcal{H}_Z$ denote the set of all valuation functions for Z.

Definition 1. *A plant is a tuple* $(\mathcal{A}, D, \widehat{P})$ *where* $\mathcal{A}$ *is an architecture (say having* k *sites* $\{s_1, \ldots, s_k\}$*),* D *is a domain definition for* $\mathcal{A}$*,* $\widehat{P}$ *is a set of* programs*, one at each site — i.e.* $\widehat{P}$ *is a tuple* $\langle P_1, \ldots P_k \rangle$*. Each* P_i *is a transition system* $(Q_i, q_i^{in}, \delta_i)$ *where* Q_i *is a set of states,* $q_i^{in} \in Q_i$ *is the initial state,* δ_i *is a non-deterministic transition function* $\delta_i : Q_i \times \mathcal{H}_{in(s_i)} \to \mathcal{P}(Q_i \times \mathcal{H}_{out(s_i)})$.[1] □

The transition function of each program defines the different ways in which a site can react to a set of inputs on its input channels, by giving the possible sets of values which can be written on the output channels together with a corresponding state change. We say a plant is *finite* if Q_i is finite for each P_i.

Let $(\mathcal{A}, D, \widehat{P})$ be a plant. For a program $P = (Q, q^{in}, \delta)$ at a site s in $\mathcal{A}$, a *local strategy for* P is a function $f : Q \times \mathcal{H}^+_{in(s)} \to Q \times \mathcal{H}_{out(s)}$ such that $\forall q \in Q$, $\pi \in \mathcal{H}^+_{in(s)}$, if $\pi = \pi' \cdot h$ then $f(q, \pi) \in \delta(q, h)$. Thus the local strategy f is an advice function for P which looks at the history of values (π') on the local input channels and the current values on them (h), and prescribes a move which the local program P should take.

A *distributed control-strategy* is a set of local strategies, one for each site: i.e. a tuple $f = \langle f_1, \ldots f_k \rangle$ where f_i is a local strategy for P_i. We call a plant along-with a strategy $((\mathcal{A}, D, P), f)$ a *controlled system*. Let us fix for now a controlled system $((\mathcal{A}, D, P), f)$.

We need some notations for talking about sequences. For a sequence α, let $\alpha[i]$ denote the i^{th} atom in α and $\alpha[i, j]$ denote the finite subsequence of α from the i^{th} to the j^{th} element, both inclusive, for $0 \leq i \leq j$, $i, j \in \mathbb{N}$. Also, we denote by $inf(\alpha)$ the set of elements which occur infinitely often in α. If α is a sequence of functions on a domain Z, let $\alpha \downarrow Z'$, where $Z' \subseteq Z$, denote the sequence of functions obtained by restricting each function in α to Z'.

Consider an environment input sequence (on the channels in X) $\alpha \in \mathcal{H}_X$. Since P, when controlled by the strategy f, is deterministic, there is a unique way in which the plant and controller respond to the external input — i.e. there is a unique sequence of states each program takes and a unique set of channel values sent along each channel. We can define these sequences as follows. Let $\beta \in (\mathcal{H}_{X \cup T})^\omega$ and $\gamma \in (Q_1 \times \ldots Q_k)^\omega$ be such that:

1. $\gamma[0] = \langle q_1^{in}, \ldots, q_k^{in} \rangle$
2. $\beta \downarrow X = \alpha$
3. $\forall t \in T$, $j \in \mathbb{N}$, if $w(t) = s_i$ and $\gamma[j] = \langle q_1, \ldots q_k \rangle$ with $f_i(q_i, \beta[0, j] \downarrow in(s_i)) = (q_i', h)$, then $\beta[j](t) = h(t)$
4. $\forall j \in \mathbb{N}$, if $\gamma[j] = \langle q_1, \ldots q_k \rangle$ and $\gamma[j+1] = \langle q_1', \ldots q_k' \rangle$, then it must be the case that $\forall i \in \{1, \ldots, k\}$: $f_i(q_i, \beta[0, j] \downarrow in(s_i)) = (q_i', h)$ for some $h \in \mathcal{H}_{out(s_i)}$

The definitions above formalize how the programs behave when they get an external input. (1) says that the global state-sequence starts with the initial states. The next condition requires that the values which the external channels take are defined by the external input α. (3) and (4) demand that the values the internal channels take and the evolution of states are according to the moves

[1] $\mathcal{P}(R)$ denotes the power-set of R

defined by the local strategy. It is easy to see that, since the architecture is acyclic, there are unique sequences β and γ which satisfy the above conditions. We call γ the *state-behaviour* of the system for the input α.

The specification is now defined on the *local* state-behaviours of the system. Since we wish to capture local linear-time properties, we have local Rabin winning conditions and the specification then demands that the local runs of the controlled system meet these conditions. We could instead work with temporal logic specifications, one at each site, which describes the behaviour of the local channels (external *and internal*) and the local states. However, since we can cast this problem in terms of Rabin conditions (by building a deterministic Rabin automaton accepting the desired behaviours [Saf88] and taking its intersection with the local plant), we can reduce this problem to our setting. coded into the states of the plant.

A *local Rabin winning condition* $\mathcal{R}_i$ for a site s_i is a set $\{(R_1, G_1), \ldots (R_m, G_m)\}$ where R_j, G_j are subsets of Q_i. A *Rabin winning condition* $\mathcal{W}$ is a tuple $\langle \mathcal{R}_1, \ldots \mathcal{R}_k \rangle$ where each $\mathcal{R}_i$ is a local Rabin winning condition for s_i. Let $\gamma \in (Q_1 \times \ldots Q_k)^\omega$ be a sequence of global states of the system. Let $\gamma \downarrow i$ denote the sequence in Q_i^ω obtained by projecting γ to the component involving Q_i. γ is said to *satisfy* a Rabin condition $\mathcal{W}$ if for each site s_i, there is a pair (R, G) in $\mathcal{R}_i$ such that $inf(\gamma \downarrow i) \cap R = \emptyset$ and $inf(\gamma \downarrow i) \cap G \neq \emptyset$.

Finally, a controlled system is said to satisfy a Rabin winning condition $\mathcal{W}$ if for every sequence of external inputs $\alpha \in (\mathcal{H}_X)^\omega$, the state-behaviour γ defined by α satisfies $\mathcal{W}$. Given a winning condition, a strategy f for a plant $(\mathcal{A}, D, P)$ is said to be *winning* if the controlled system $((\mathcal{A}, D, P), f)$ satisfies the winning condition. Now, the control-synthesis problem for an architecture $\mathcal{A}$ is: Given a finite plant $(\mathcal{A}, D, P)$, and a Rabin winning condition, does there exist a winning strategy for the plant?

An important class of architectures are the class of *clean* pipelines which are pipelines that have external inputs only at the two endpoints (see Fig. 1): An architecture $\mathcal{A}$ is said to be a *clean pipeline* if it has k sites $s_1, \ldots s_k$ (for some $k \in \mathbb{N}$), two external input channels x_1 and x_2, $k-1$ internal channels $t_1, \ldots t_{k-1}$, with $r(x_1) = s_1$, $r(x_2) = s_k$, $w(t_i) = s_i$ and $r(t_i) = s_{i+1}$, for $i \in \{1, \ldots, k-1\}$.

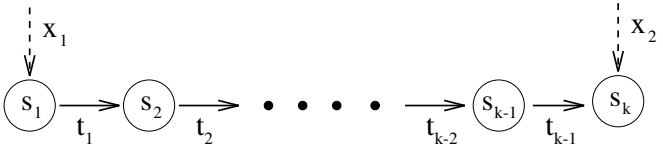


Fig. 1. A generic clean pipeline

We also need the notion of a sub-architecture — an architecture $\mathcal{A}'$ is a sub-architecture of an architecture $\mathcal{A}$ if the graph of $\mathcal{A}'$ is isomorphic to a subgraph

of the graph of $\mathcal{A}$. Note that an architecture is a sub-architecture of itself. We can now state the main result of the paper.

Theorem 1. *Let $\mathcal{A}$ be an architecture. The control-synthesis problem for $\mathcal{A}$ is decidable iff each connected component of $\mathcal{A}$ is a sub-architecture of a clean pipeline.* □

3 Decidable Architectures

In this section, we show that architectures where each connected component is a sub-architecture of a clean pipeline is decidable. Since we have local winning conditions, it is easy to observe that the control-synthesis problem for an architecture $\mathcal{A}$ is decidable iff the control-synthesis problem is decidable for each of its connected components. Hence it suffices to prove that the problem is decidable for architectures which are sub-architectures of clean pipelines.

We use alternating and non-deterministic tree automata to prove our decidability results. Due to lack of space, we assume a standard presentation of these automata [Tho90, MS95].

A *tree* is a directed acyclic graph $T = (V, E)$ which has a designated *root* r which does not have a parent. Every other node is reachable from r and has a unique parent. We say v' is a child of v if $(v, v') \in E$. For a set Σ, a Σ-labelled tree is a pair (T, τ) where $T = (V, E)$ is a tree and $\tau : V \to \Sigma$. Let Υ be a finite set. Then Υ^* can be viewed as a tree $T_\Upsilon = (\Upsilon^*, E)$ where $(x, x \cdot d) \in E$, for every x in Υ^* and d in Υ. We refer to this as the Υ-tree.

Consider a plant $(\mathcal{A}, D, P)$ and a distributed control-strategy f for it. Let s be a site with an output channel t in $\mathcal{A}$. Let $\mathcal{L} \subseteq D_t^\omega$ be the language of infinite strings output on t (by considering all possible inputs on the external input channels of the plant). We call such a language of infinite words, *a communication language for the channel t*. Let $Pref(\mathcal{L}) = \{x \mid \exists \beta \in \mathcal{L}, x \text{ is a prefix of } \beta\}$. Then it is not difficult to see that $\mathcal{L} = lim(Pref(\mathcal{L}))$ where $lim(L) = \{\alpha \in \Sigma^\omega \mid \text{for every prefix } x \text{ of } \alpha, x \in L\}$. Also, $\mathcal{L} \neq \emptyset$.

So $L = Pref(\mathcal{L}) \subseteq D_t^*$, the set of finite sequences sent along t represents the set of infinite sequences sent along the channel as well. So $\mathcal{L}$ can be represented (uniquely) by a $\{\top, \bot\}$-labelled D_t-tree $T = (D_t^*, \tau)$, where $\tau(x) = \top$ if $x \in L$ and $\tau(x) = \bot$, otherwise. In such a tree if a node has label $\top$ then it will have at least one child with the label $\top$ and if a node has label $\bot$ then all its children will have the label $\bot$. Also, the root, ϵ is labelled $\top$. Clearly each such $\{\top, \bot\}$-labelled D_t-tree (which we call t-type trees) uniquely represents a communication language of the channel t and our automata run over such trees. If T is a t-type tree then we let $Lang(T)$ denote the language of infinite strings it represents.

Let us fix a clean pipeline which has k sites, as shown in Figure 1. We refer to s_1 as the *left-site*, s_k as the *right-site* and each of the s_i's, $1 < i < k$ as *middle sites*.

Let s be the left-site of a clean pipeline with input channel x and output channel t. Suppose P is the program at s, and there is is a local *winning* strategy f at s and $\mathcal{L}$ is the language of infinite words sent along t as a result of the pair

(P, f) reacting to all possible inputs on x. Then $\mathcal{L}$ is said to be an *s-successful* language. For a right-site s, with input channels t and x, we say that a language $\mathcal{L}$ of infinite strings over D_t is s-successful, if there is a strategy f at s which can work on these inputs, and arbitrary inputs on x, and win locally. For a middle-site s with input channel t and output channel t', we say that $\mathcal{L}' \subseteq D(t')^\omega$ is successfully generable by s on $\mathcal{L} \subseteq D(t)^\omega$ if there is a strategy at s which wins locally on reading inputs from $\mathcal{L}$ on t and produces $\mathcal{L}'$ on t'.

Lemma 1. *Let s be the left-site of a clean pipeline with input channel x, output channel t and program P. Then there is an alternating tree automaton (on t-type trees) which accepts a t-type tree T iff the language represented by T has an s-successful sublanguage.*

Proof: The automaton we construct, while running over a t-type tree T, guesses a local strategy f for the program at s, makes sure that f produces no string which is not in $Lang(T)$ and also checks that f is locally winning.

The automaton has, in its state-space, a component encoding which state of the program it is currently simulating. Then reading a node y of T, it does the following:

- Guess a set of moves from the current state for each possible input d in $D(x)$.
- The automaton propagates, for each $d \in D(x)$, a copy along the direction $d' \in D(t)$ where d' is the output of the plant on d according to the guessed move. The corresponding successor state of the program is also propagated and the automaton will check in these copies whether the labels of the nodes it reads are $\top$. This will ensure that the outputs are allowed by T.

The acceptance condition ensures that paths on a run encode state-sequences which satisfy the local winning condition of P. Since each node in a run corresponds to a unique input history, the independent guessing of the strategy at these nodes is justified. □

Note that the automaton accepts a tree provided the language represented by the tree merely *contains* an s-successful language. It seems hard to strengthen this containment to equality. However, the present version will suffice.

Lemma 2. *Let s be a right-site of a pipeline with $in(s) = \{x, t\}$ and let the program at s be P. Then there is an alternating tree automaton on t-type trees which accepts a tree T iff the language that T represents is s-successful.*

Proof: The automaton guesses a local strategy for P at s on input sequences $\alpha \in Lang(T)$ along t and arbitrary input sequences $\beta \in D(x)^\omega$ on x and makes sure that f is winning for all local runs on these sequences.

The automaton keeps track in its state-space the current state of P it is simulating. Reading a node y of the input tree, it does the following:

- Guess $Y \subseteq D(t)$ corresponding to the set of successors of y labelled $\top$. The automaton will (in its next move) check if Y is indeed the set of $\top$-successors.
- The strategy has to handle all inputs in Y on the channel t along with an arbitrary input in $D(x)$ on channel x. The automaton guesses such a strategy at this point by guessing moves from the current state of P on each

$h \in \mathcal{H}_{\{x,t\}}$ with $h(t) \in Y$. It then propagates along each direction d in Y, a copy of the automaton for each $d' \in D(x)$ corresponding to the chosen move when channel t carries d and channel x carries d'. It propagates the corresponding state of P as well.

The acceptance condition is that all paths on a run encode a state-sequence in P which satisfies the local winning condition of P. Again, since each node in a run corresponds to a unique input history, the guessing of the strategy at these points independently is justified. □

Theorem 2. *The two-site clean pipeline is decidable.*

Proof: Let the sites and channels of the pipeline be labelled as in Figure 1. Using Lemma 1, construct an automaton A_1 which accepts a t_1-type tree T iff s_1 can successfully generate a sublanguage of $Lang(T)$. Using Lemma 2, construct A_2 which accepts t_1-type trees which represent languages which s_2 can win on. The claim now is that a distributed winning strategy exists iff $L(A_1) \cap L(A_2)$ is nonempty.

Assume $T \in L(A_1) \cap L(A_2)$ and let $\mathcal{L}$ be the language it represents. Then there is a strategy f_2 at s_2 which wins on $\mathcal{L}$. Also, there is a local winning strategy f_1 at S_1 which generates a sublanguage $\mathcal{L}'$ of $\mathcal{L}$. However, since the local winning conditions are linear-time specifications, f_2 wins on $\mathcal{L}'$ as well. Hence $\langle f_1, f_2 \rangle$ is a distributed winning strategy. Furthermore, one can construct, from the runs of A_1 and A_2 on a regular tree in $L(A_1) \cap L(A_2)$, a strategy which can be realized as a finite-state transition system. Also, it is easy to see that if $\langle f_1, f_2 \rangle$ is any winning distributed strategy, then the tree corresponding to the language f_1 generates is accepted by A_1 as well as A_2. □

Lemma 3. *Let s be a middle-site of a clean pipeline with $in(s) = \{t\}$ and $out(s) = \{t'\}$, and let the program at s be P. Let A be a nondeterministic automaton accepting t-type trees. Then there is an automaton on $\{\top, \bot\}$-labelled t'-type trees that accepts a tree T' iff there is a t-type tree T accepted by A and a language $\mathcal{L}_0 \subseteq Lang(T')$ such that $\mathcal{L}_0$ is successfully generable by s on $Lang(T)$.*

Proof: Let T' be an input to the automaton and $\mathcal{L}'$ be the language it represents. The automaton, while reading T', guesses a t-type tree T, guesses a run of A on T, guesses a strategy f on the strings represented in T and makes sure that the run on T is accepting, makes sure that the strategy outputs strings which are included in $\mathcal{L}'$ and makes sure that the strategy locally wins.

A node in the run on T' corresponds to a node y' in T' as well as a node x of the tree T being guessed — here x is the sequence in $D(t)^*$ on which the guessed strategy has output y'. Note that each sequence in $D(t)^*$ can lead to at most one sequence in $D(t')^*$ being output and hence guessing of the tree T at nodes of the run is justified.[2]

The state-space of the automaton has both the current state of P as well as a state of the automaton A which represents the state-label of the corresponding node in T in the guessed run on T. The automaton at a node in the run corresponding to the node y' in T' and x in T does the following:

[2] If the site also has an external input, this will not be the case.

- Guess the set $Y' \subseteq D(t')$ which corresponds to the children of y' labelled $\top$.
- Guess the labels of the children of x in T. This is the point where T is being guessed. Let $X \subseteq D(t)$ be the children of x labelled $\top$.
- The automaton now guesses a move of P from the current state on each $d \in X$ and makes sure that the output on t is in Y'. It then propagates along each direction $d' \in Y'$ in T, many copies of itself — each corresponding to a $d \in D(t)$ on which the guessed move outputs d'. The appropriate successor state of P is propagated. The automaton also guesses a transition of A from the node x and propagates these automaton states as well.

The acceptance condition makes sure that along any path in the run, the state-sequence of P meets the local winning condition of P and the state-sequence of the automaton meets the winning condition of A. □

Theorem 3. *The control-synthesis problem for clean pipelines is decidable.*

Proof: We start with the left-site of the pipeline, use Lemma 1 and walk down the pipeline by successively using Lemma 3. After each site, we have to convert the alternating automata we get to a nondeterministic one [MS95] so as to apply Lemma 3. In the end, we intersect the automata we have got with that obtained using Lemma 2. Then by an argument similar to the one in Theorem 2, we can show that there is a nonempty intersection iff there is a distributed controller and if a controller exists, we can synthesize one which is finite-state. □

The results imply the decidability of a related realizability problem: given a clean pipeline and localised temporal logic specifications at each site, where each specification can express properties of the local channels at that site, is there a program at each site such that the combined behaviour meets the specification? This problem can be reduced to the control-synthesis problem by choosing a trivial plant at each site which permits all possible ways of writing onto the local output channels.

4 Undecidable Architectures

We show now that any architecture which is not a sub-architecture of a clean pipeline is undecidable. We show first the undecidability of the three basic architectures in Figure 2. The reductions are from the halting problem for deterministic Turing machines starting with blank tapes. Our proofs are extensions of the the undecidability proof developed in [PR90].

A configuration of a deterministic Turing machine M is a sequence $C \in \Gamma^* \cdot Q \cdot \Gamma^+$ where Γ is the set of tape symbols and Q is the set of states. If $C = xqy$, with $q \in Q$, then the machine has $x \cdot y$ written on the tape with the head position on the cell after x. The initial configuration, $C_{in} = q_{in} \cdot \flat$ where q_{in} is the initial state and $\flat$ is the special tape symbol called blank. The transition relation $\vdash$ on configurations is defined in the obvious way. We say that the machine halts on the blank-tape if $C_{in} \vdash^* C_h$ where the state in C_h is q_h, a designated halt state.

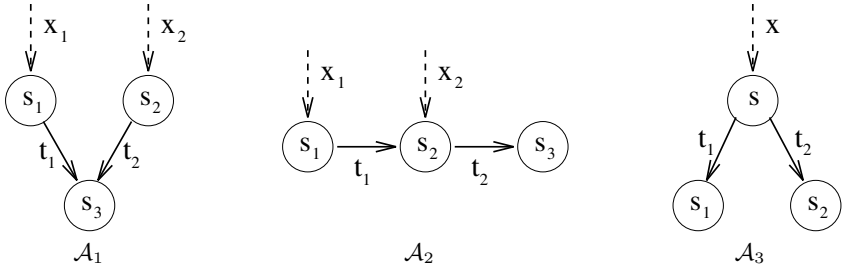


Fig. 2. Basic undecidable architectures

The sites host programs which have associated strategies to output configurations. The (finite state) program outputs words in $\Gamma^* \cdot Q \cdot \Gamma^+$ with the strategy deciding which configurations are output. The input channels carry two symbols S (Start outputting a new configuration) and N(output the next symbol of the current configuration). On receiving S, a program will output \$, followed by a configuration while reading N and finish the configuration by outputting \$. It waits while reading N, and outputs a special symbol $*$ during this time, till it get an S on which it starts outputting another configuration. The first configuration output by the program is always C_{in}.

Lemma 4. *The control-synthesis problem for the architecture $\mathcal{A}_1$ is undecidable.*

Proof: The main idea of the proof is to make s_1 and s_2 always send their current states to s_3. Site s_3 now has the global view of s_1 and s_2 and a global specification of s_1 and s_2 (exploited in [PR90] to get the undecidability argument) can be stated as a local specification for s_3. □

Lemma 5. *The control-synthesis problem for the architecture $\mathcal{A}_2$ is undecidable.*

Proof: Site s_1 will output configurations when prompted by the environment through channel x_1. Site s_3 will, when prompted by s_2 on t_2, "accept" configurations instead of outputting them; when it starts a configuration, it will generate it one unit time in advance and keep the generated symbol of $\Gamma \cup Q$ in its state-space. It proceeds from this state only if the input it receives on t_2 is the same as the symbol it has committed to. It then proceeds to commit the next symbol. This can be looked upon as s_3 generating configurations which s_2 has to predict correctly.

Site s_2 can go into two modes, A and B, the decision being taken according to the first environment input on x_2. In Mode A, the program at s_2 simply passes the configurations which it receives on t_1 to t_2. In Mode B, the program first outputs the initial configuration to s_3 and after that, each time it receives a configuration C on t, it propagates online C' to t_2 where $C \vdash C'$.

If s_3 receives a symbol it has not committed to, it goes to a reject state. Mode A ensures that the two sites output/accept the same configuration sequences while Mode B ensures that if the i^{th} configuration output by s_1 is C and the $(i+1)^{th}$ configuration accepted by s_2 is C', then $C \vdash C'$. So the only way the

plant can hope to win is by s_1 and s_3 accepting the configuration sequence of M. By introducing a winning condition on s_2 which makes sure that s_2 locally wins only if it outputs the halting configuration, one can show that the plant has a distributed winning strategy iff M halts on the blank tape. □

Lemma 6. *The control-synthesis problem for the architecture $\mathcal{A}_3$ is undecidable.*

Proof: As done by s_3 of $\mathcal{A}_2$ in the previous lemma, s_1 and s_2 will now accept configuration of M. Site s can be in two modes, A and B, the mode chosen by the first input on x. In Mode A, the program at s_1 passes the initial configuration C_{in} to s_1 and makes s_2 wait. Then, while getting as input an arbitrary configuration C from the environment on x, it passes C to s_2 and simultaneously passes C' to s_1 where $C \vdash C'$. Mode B is analogous with the roles of s_1 and s_2 interchanged.

To force s_1 and s_2 to accept the correct configuration sequence of M, we would like the environment to win iff it can get the site scheduled first to be unstuck and get the other stuck. The trick is to have another mode C for s where the plant is forced to emulate the combined (product) behaviour of s_1 and s_2. The winning condition can now be stated on the state-space of s. Then one can make make sure that one of the sites, say s_1, wins when it accepts the halting configuration. One can show now that a distributed controller exists iff M halts on the blank tape. □

Using Lemma 4 we can show that any architecture which has a site s with two internal input channels is undecidable. The idea is to pick the minimal sites above the two internal channels for s and make the rest of the sites "dummy" by making them just pass their input to their output and always win locally. We can then reduce the control-synthesis problem of $\mathcal{A}_1$ to that over this architecture. Similarly, using Lemma 6 we can show that any architecture which has a site with two internal output channels is undecidable.

What we are left with are pipelines. Since we require each process to have an input channel, the left-site of the pipeline must have an external input channel. Consider a pipeline (which is not a clean pipeline) with sites $\{s'_1, \ldots, s'_k\}$, $k > 2$, with s'_i having an external input channel where $1 < i < k$. We can reduce the control-synthesis problem for $\mathcal{A}_2$ to the control-synthesis problem for this pipeline, by coding the program at s_1 into s'_1, the program at s_2 into s'_i and the program at s_3 into s'_k. The remaining sites of the pipeline will be "dummy". Hence we have:

Theorem 4. *If $\mathcal{A}$ is an architecture which has a connected component which is not a sub-architecture of a clean pipeline, then the control-synthesis problem for $\mathcal{A}$ is undecidable.* □

The results above can be suitably changed to show that even for weaker winning conditions such as Büchi, co-Büchi, or even safety conditions, the architectures remain undecidable.

Acknowledgement. We would like to thank Wolfgang Thomas and Christof Löding for several fruitful discussions on early drafts of this paper.

References

[AHK97] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proc. 38th IEEE FOCS*, pages 100–109, October 1997.

[ALW89] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 16th ICALP*, vol. 372, pages 1–17. LNCS, Springer-Verlag, July 1989.

[BL69] J.R. Büchi and L.HG. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138:295–311, 1969.

[dAH00] Luca de Alfaro and Thomas A. Henzinger. Concurrent omega-regular games. In *Proc., LICS '00, 15th Annual Conf.*, pages 141–154, 2000.

[dAHM00] Luca de Alfaro, Thomas A. Henzinger, and F.Y.C. Mang. The control of synchronous systems. In *Proc., CONCUR '00, 11th Int. Conf.*, vol. 1877 of *LNCS*, pages 458–473, Penn. State Univ, USA, Sept. 2000.

[KMTV00] O. Kupferman, P. Madhusudan, P.S. Thiagarajan, and M. Vardi. Open systems in reactive environments: Control and synthesis. In *Proc., CONCUR '00, 11th Int. Conf.*, vol. 1877 of *LNCS*, USA, Sept. 2000.

[KV96] O. Kupferman and M.Y. Vardi. Module checking. In *CAV, Proc. 8th Intl. Conf.*, vol. 1102 of *LNCS*, pages 75–86. Springer-Verlag, 1996.

[KV99] O. Kupferman and M.Y. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2):245 – 263, June 1999.

[MS95] D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.

[MT98] P. Madhusudan and P. S. Thiagarajan. Controllers for discrete event systems via morphisms. In *In Proc., CONCUR'98, 9th Int. Conf.*, vol. 1466 of *LNCS*, pages. 18–33, France, September 1998.

[PR79] G.L. Peterson and J.H. Reif. Multiple-person alternation. In *Proc. 20th IEEE Symp. on FOCS*, pages 348–363, 1979.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Prog. Languages*, Austin, January 1989.

[PR90] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. 31st IEEE Symp. FOCS*, pages 746–757, 1990.

[Rab72] M.O. Rabin. Automata on infinite objects and Church's problem. *Amer. Mathematical Society*, 1972.

[RW89] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *IEEE Transactions on Control Theory*, 77:81–98, 1989.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. FOCS*, pages 319–327, White Plains, October 1988.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 165–191, 1990.

[Tho95] W. Thomas. On the synthesis of strategies in infinite games. In E.W. Mayr and C. Puech, editors, *Proc. 12th Symp. on Theoretical Aspects of Comp. Sc.*, vol. 900 of *LNCS*, pages 1–13. Springer-Verlag, 1995.

[MT01] P. Madhusudan and P. S. Thiagarajan. Distributed Controller Synthesis for Local Specifications. Technical Report TCS-01-1, *Chennai Mathematical Institute*, India. Available at `http://www.smi.ernet.in`

A Distributed Abstract Machine for Safe Ambients

(Extended Abstract)

Davide Sangiorgi[1] and Andrea Valente[2]

[1] INRIA Sophia-Antipolis, France. davide.sangiorgi@inria.fr
[2] Università di Torino, Italy. valente@di.unito.it

1 Introduction

The Ambient calculus [4] is a model for mobile distributed computing. An ambient is the unit of movement. Processes within the same ambient may exchange messages; ambients may be nested, so to form a hierarchical structure. The three primitives for movement allow: an ambient to enter another ambient, $n[\,\mathtt{in}\,m.\,P \mid Q\,] \mid m[\,R\,] \longrightarrow m[\,n[\,P \mid Q\,] \mid R\,]$; an ambient to exit another ambient, $m[\,n[\,\mathtt{out}\,m.\,P \mid Q\,] \mid R\,] \longrightarrow n[\,P \mid Q\,] \mid m[\,R\,]$; a process to dissolve an ambient boundary thus obtaining access to its content, $\mathtt{open}\,n.\,P \mid n[\,Q\,] \longrightarrow P \mid Q$.

Several studies of the basic theory of the Ambient calculus have recently appeared, concerning for instance behavioural equivalences, types, logics, static analysis techniques [5,6,1,7,12]. In comparison, little attention has been given to implementations. The only implementations of Ambients we are aware of are Cardelli's [2,3], and Fournet, Lévy and Schmitt's [9]. The latter, formalised as a translation of Ambients into the distributed Join Calculus, is the only distributed implementation. Although ingenious, the algorithms that these implementations use for simulating the ambient reductions are fairly complex.

One of the difficulties of a distributed implementation of an ambient-like language is that each movement operation involves ambients on different hierarchical levels. For instance, the ambients affected by an `out` operation are the moving ambient, and its initial and its final parent; at the beginning they reside on three different levels. In [2,3] locks are used to achieve a synchronisation among all ambients affected by a movement. In a distributed setting, however, this lock-based policy can be expensive. For instance, the serialisations introduced diminish the parallelism of the whole system. In [9] the synchronisations are simulated by means of protocols of asynchronous messages. The problems of implementation have been a restraint to the development of programming languages based on Ambients and to experimentation of Ambients on concrete examples. In our opinion, implementation is one of the aspects of Ambients that most need investigations.

In this paper we study an abstract machine for a distributed implementation of an ambient-like calculus. The algorithms of our abstract machine are quite different from, and simpler than, those of [2,3,9], mainly for two reasons. The first – the most important — is that the calculus that we actually take is typed Safe Ambients [11] (SA) rather than untyped Ambients. SA is a variant of the

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 408–420, 2001.

original calculus that eliminates certain forms of interference in ambients, the grave interferences. They are produced when an ambient tries to perform two different movement operations at the same time, as for instance $n[\,\mathtt{in}\,h.\,P \mid \mathtt{out}\,n.\,Q \mid R\,]$. The control of mobility is obtained in SA by a modification of the syntax and a type system. In [11] the absence of grave interferences is used to develop an algebraic theory and prove the correctness of some examples. One of the contributions of this paper is to show that the absence of grave interferences also brings benefits in implementations.

The second reason for the differences in our abstract machine is the separation between the logical structure of an ambient system and its physical distribution. Exploiting this, the interpretation of the movement associated to the capabilities is reversed: the movement of the `open` capability is physical, that is, the location of some processes changes, whereas that of `in` and `out` is only logical, that is, some hierarchical dependencies among ambients may change, but not their physical location. Intuitively, `in` and `out` are acquisition of access rights, and `open` is exercise of them.

The differences also show up in the correctness proof of the abstract machine, which is much simpler than the correctness proof of the Join implementation.

Of course another difference is that our algorithms are formulated as an abstract machine. The machine is independent of a specific implementation language, and can thus be used as a basis for implementations on different languages. In the paper we sketch one such implementation, written in Java.

2 Safe Ambients: Syntax and Semantics

We briefly describe typed Safe Ambient (SA), from [11]. In the reduction rules of the original Ambient calculus, mentioned in Section 1, an ambient may enter, exit, or open another ambient. The second ambient undergoes the action; it has no control on *when* the action takes place. In SA this is rectified: *coactions* $\overline{\mathtt{in}}\,n, \overline{\mathtt{out}}\,n, \overline{\mathtt{open}}\,n$ are introduced with which any movement takes place only if both participants agree. The syntax of SA is the following, where $n, m, \ldots$ are *names*, $x, y, \ldots$ are *variables*, $X, Y, \ldots$ are *recursion variables*:

$$M, N := x \mid n \mid \mathtt{in}\,M \mid \overline{\mathtt{in}}\,M \mid \mathtt{out}\,M \mid \overline{\mathtt{out}}\,M \mid \mathtt{open}\,M \mid \overline{\mathtt{open}}\,M$$

$$P, Q, R := \mathbf{0} \mid P \mid Q \mid (\boldsymbol{\nu} n)\,P \mid M.\,P \mid M[\,P\,] \mid \langle M \rangle \mid (x)P \mid X \mid \mathtt{rec}\,X.\,P$$

Expressions that are not variables or names are the *capabilities*. We often omit the trailing $\mathbf{0}$ in processes $M.\,\mathbf{0}$. Parallel composition has the least syntactic precedence, thus $m[\,M.\,P \mid Q\,]$ reads $m[\,(M.\,P) \mid Q\,]$. An ambient, or a parallel composition, or variable, is *unguarded* if it is not underneath a capability or an abstraction. In a recursion $\mathtt{rec}\,X.\,P$, the recursion variable X should be guarded in P. For simplicity of presentation we omit path expressions in the syntax.

Below are the reduction axioms: those for movement, and the communication rule (communication is asynchronous, takes place inside ambients, and is anonymous—it does not use channel or process names):

$$n[\,\mathtt{in}\,m.\,P_1 \mid P_2\,] \mid m[\,\overline{\mathtt{in}}\,m.\,Q_1 \mid Q_2\,] \longrightarrow m[\,n[\,P_1 \mid P_2\,] \mid Q_1 \mid Q_2\,] \qquad \text{[R-IN]}$$

$$m[\,n[\,\mathtt{out}\,m.\,P_1 \mid P_2\,] \mid \overline{\mathtt{out}}\,m.\,Q_1 \mid Q_2\,] \longrightarrow n[\,P_1 \mid P_2\,] \mid m[\,Q_1 \mid Q_2\,] \qquad \text{[R-OUT]}$$

$$\mathtt{open}\,n.\,P \mid n[\,\overline{\mathtt{open}}\,n.\,Q_1 \mid Q_2\,] \longrightarrow P \mid Q_1 \mid Q_2 \qquad \text{[R-OPEN]}$$

$$\langle M\rangle \mid (x)P \longrightarrow P\{^{M}/_{x}\} \qquad \text{[R-MSG]}$$

The inference rules allow a reduction to occur underneath a restriction, a parallel composition, and inside an ambient. Moreover, the structural congruence relation ($\equiv$) can be applied before a reduction step. Structural congruence is used to bring the participants of a potential interaction into contiguous positions; its definition is standard, and includes rules for commuting the positions of parallel components, for stretching the scope of a restriction, for unfolding recursions. We write $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$. The use of coactions, in the syntax and operational rules, is the only difference between (untyped) SA and and the original Ambient calculus.

Up to structural congruence, every ambient in a term can be rewritten into a normal form

$$n[\,P_1 \mid \ldots \mid P_s \mid m_1[\,Q_1\,] \mid \ldots \mid m_r[\,Q_r\,]\,]$$

where P_i $(i = 1..s)$ does not contain unguarded ambients or unguarded parallel compositions. In this case, $P_1, \ldots, P_s$ are the *local processes* of the ambient, and $m_1[\,Q_1\,] \mid \ldots \mid m_r[\,Q_r\,]$ are the *subambients*.

SA has two main kinds of types: *single-threaded* and *immobile*. We consider them separately. We begin with the single-threaded types, which we informally describe below. We consider immobility types in Section 6.

The capabilities of the local processes of an ambient control the activities of that ambient. In an untyped (or immobile) ambient such control is distributed over the local processes: any of them may exercise a capability. In a *single-threaded* (ST) ambient, by contrast, at any moment at most *one* process has the control thread, and may therefore use a capability. An ST ambient n is willing to engage in at most one interaction at a time with external or internal ambients. Inside n, however, several activities may take place concurrently: for instance, a subambient may reduce, or two subambients may interact with each other. Thus, if an ambient n is ST, the following situation, where at least two local processes are ready to execute a capability, cannot occur: $n[\,\mathtt{in}\,m.\,P \mid \mathtt{out}\,h.\,Q \mid R\,]$. The control thread may move between processes local to an ST ambient by means of an open action. Consider, for instance, a reduction $n[\,\mathtt{open}\,m.\,P \mid m[\,\overline{\mathtt{open}}\,m.\,Q\,]\,] \longrightarrow n[\,P \mid Q\,]$ where n and m are ST ambients. Initially $\mathtt{open}\,m.\,P$ has the control thread over n, and $\overline{\mathtt{open}}\,m.\,Q$ over m. At the end, m has disappeared; the control thread over n may or may not have moved from P to Q, depending on the type of m. If the movement occurs, Q can immediately exercise a capability, whereas P cannot; to use further capabilities within n, P will have to get the thread back.

For simplicity, we assume here a strong notion of ST, whereby a value message $\langle M\rangle$ never carries the thread. In [11] a weaker notion is used, where also messages may carry the thread. In the remainder, all processes are assumed to be well-typed, and closed (i.e., without free variables).

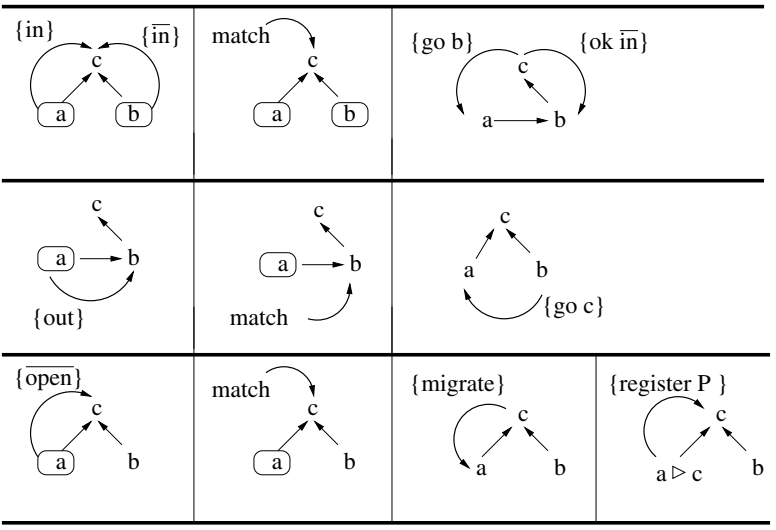


Fig. 1. The simulation of the reductions R-IN, R-OUT, and R-OPEN in PAN

3 The Abstract Machine, Informally

We describe the data structures and the algorithms of the abstract machine, called PAN. PAN separates between the logical and the physical distribution of the ambients. The logical distribution is given by the tree structure of the ambient syntax. The physical distribution is given by the association of a location to each ambient.

In PAN, an ambient named n is represented as a *located ambient* $h{:}\, n[\, P\,]_k$, where h is the location, or site, at which the ambient runs, k is the location of the parent of the ambient, and P collects the processes local to the ambient. While the same name may be assigned to several ambients, a location univocally identifies an ambient; it can be thought of as its physical address.

A tree of ambients is rendered, in PAN, by the parallel composition of the (unguarded) ambients in the tree. In this sense, the physical and the logical topology are separated: the space of physical locations is flat, and each location hosts at most one ambient, but each ambient knows the location at which its parent resides. For instance, an SA term $n[\, P_1 \mid P_2 \mid m_1[\, Q_1\,] \mid m_2[\, Q_2\,]\,]$, where P_1 and P_2 are the local processes of n, and Q_i $(i = 1, 2)$ is a local process of m_i (i.e., m_i has no subambients), becomes in PAN:

$$h{:}\, n[\, P_1 \mid P_2\,]_{\texttt{root}} \parallel k_1{:}\, m_1[\, Q_1\,]_h \parallel k_2{:}\, m_2[\, Q_2\,]_h$$

where h, k_1, k_2 are different location names, `root` is a special name indicating the outermost location, and $\parallel$ is parallel composition of located ambients. (The above configuration is actually obtained after two creation steps, in which the root ambient spawns off the two ambients located at k_1 and k_2.) Since ambients may run at different physical sites, they communicate with each other by means of *asynchronous* messages.

All the actions (`in` , `out` , and `open`) can modify the logical distribution. Only `open` , however, can modify the physical distribution. The algorithms that PAN adopts to model reduction in SA are based on 3 steps: first, a *request* message is sent upward, from a child ambient that wants to move (logically or physically) to its parent; second, a *match* is detected by the parent itself; third, a *completion* message is sent back to the child, for its relocation. The only exception is the algorithm for `open` , where a further message is needed to migrate the child's local processes to the parent. These steps are sketched in Figure 1, where a, b, c represent three ambients, a straight line represents a pointer from an ambient to its parent, and a curved line represents the sending of a message. Thus in first row of Figure 1, at the beginning a and b are sibling ambients and c is their parent. This figure illustrates an R-IN reduction in which a becomes a child of b. In the first phase, a demands to enter b (precisely, if n is the name of b, then a demands of entering an ambient with name n), and b accepts an ambient in. For this, a and b send requests `in` and $\overline{\texttt{in}}$ to their parent c (the actual messages may also contain the name and location of the sender; these are not shown in the figures). In the second phase, c sees that two matching requests have been sent and authorises the movement. Finally, in the third phase, c sends completion messages to a and b. The message sent to a also contains the location of b, which a will use to update its parent field. An ambient that has sent a request to its parent but has not yet received an acknowledgement back, goes into a *wait* state, in which it will not send further requests. In the figures, this situation is represented by a circle that encloses the ambient. An ambient in a *wait* state, however, can still receive and answer requests from its children and can perform local communications.

The second row of Figure 1 sketches an R-OUT reduction. In the first phase, ambient a demands its parent b to exit. When b authorises the movement (phase 2), it sends a an acknowledgement containing the location of the parent of b, namely c, and upon receiving this message (phase 3) a updates its parent field. The grandparent ambient c is not affected by the dialog between a and b. The third row of Figure 1 sketches an R-OPEN reduction. Ambient a accepts to be opened, and thus notifies its parent c. If a matching capability exists, that is, one of the processes local to c demands to open a, then c authorises a to migrate its local processes into c. Ambient a then becomes a forwarder ($a \triangleright c$ in the figure) whose job is just to forward any messages sent to a on to c. Such a forwarder is necessary, in general, because a may have subambients, which would run at different locations and which would send their requests of movement to a.

Using R-OPEN, rather than R-IN or R-OUT, for the physical movements may appear counterintuitive. One should however bear in mind that, in an ambient-like formalism, entering and exiting ambients is not very useful without opening some ambients.

4 The Abstract Machine, Formally

Syntax. The syntax of PAN is shown in Table 1. A term of PAN, a *net*, is the parallel composition of *agents* and *messages*, with some names possibly restricted. An agent can be a *located ambient* or a *forwarder*. Located ambients are

Table 1. The syntax of PAN

$a, b, .. \in \mathit{Names}$ $\quad h, k, .. \in \mathit{Locations}$ $\quad p, q, .. \in \mathit{Names} \cup \mathit{Locations}$

		Nets				*Agents*
A	$:=$	$\mathbf{0}$	(empty)	*Agent*	$:=$	$h \rhd k$ (forwarder)
	$\mid$	*Agent*	(agent)		$\mid$	$h{:}\,n[\,P\,]_k$ (located ambient)
	$\mid$	$h\{\mathit{MsgBody}\}$	(message)			
	$\mid$	$A_1 \parallel A_2$	(composition)			*Message body*
	$\mid$	$(\boldsymbol{\nu} p\,)A$	(restriction)	*MsgBody*	$:=$	*Request* (request)
					$\mid$	*Completion* (completion)

Request	$:=$	$\mathtt{in}\, n, h$	(the agent at h wants to enter n)
	$\mid$	$\overline{\mathtt{in}}\, n, h$	(the agent at h, named n, accepts someone in)
	$\mid$	$\mathtt{out}\, n, h$	(the agent at h wants to go out of n)
	$\mid$	$\overline{\mathtt{open}}\, n, h$	(the agent at h, named n, accepts to be opened)
Completion	$:=$	$\mathtt{go}\, h$	(change the parent to be h)
	$\mid$	$\mathtt{OK}\overline{\mathtt{in}}$	(request $\overline{\mathtt{in}}$ accepted)
	$\mid$	$\mathtt{migrate}$	(request $\overline{\mathtt{open}}$ accepted)
	$\mid$	$\mathtt{register}\, P$	(add P to the local processes)

Process-related syntax:

P	$:=$	$\mathbf{0}$	$\mid$	$(x)\,P$	M	$:=$	x	$\mid$	$\mathtt{out}\, M$
	$\mid$	$P_1 \mid P_2$	$\mid$	X		$\mid$	n	$\mid$	$\overline{\mathtt{out}}\, M$
	$\mid$	$(\boldsymbol{\nu} n\,)P$	$\mid$	$\mathtt{rec}\, X.\, P$		$\mid$	$\mathtt{in}\, M$	$\mid$	$\mathtt{open}\, M$
	$\mid$	$M.\, P$	$\mid$	$\mathtt{wait}.\, P$		$\mid$	$\overline{\mathtt{in}}\, M$	$\mid$	$\overline{\mathtt{open}}\, M$
	$\mid$	$M[\,P\,]$	$\mid$	$\{\mathit{Request}\}$					
	$\mid$	$\langle M \rangle$							

the basic unit of PAN, and represent ambients of SA with their local processes. The syntax of the processes inside located ambients is similar to that of processes in SA. The only additions are: the prefix $\mathtt{wait}.\, P$, which appears in an ambient when this has sent a request to its parent but has not received an answer yet; and the requests, which represent messages received from the children and not yet served. We use A to range over nets.

Semantics. The reduction relation of PAN, $\longmapsto$, from nets to nets, is defined by the rules below. The rules for local reductions, and the associated inference rule PAR-PROC, have a special format. We write $P \xrightarrow[h:n]{k} Q \gg \widetilde{Msg}$ to mean a

process P, local to an ambient n that is located at h, and whose parent is located at k, becomes Q and, as a side effect, the messages in $\widetilde{Msg}$ are generated. We use $\widetilde{Msg}$ to indicate a possibly empty parallel composition of messages. For instance, if $P \xrightarrow[h:n]{k} Q \gg \widetilde{Msg}$, then, using PROC-AGENT and PAR-AGENT, we have, for any net A:

$$A \parallel h{:}\, n[\, P\,]_k \longmapsto A \parallel h{:}\, n[\, Q\,]_k \parallel \widetilde{Msg}$$

When n or h or k are unimportant, we replace them with $-$, as in $P \xrightarrow[-:n]{k} Q \gg \widetilde{Msg}$. The rule STRUCT-CONG make use of the structural congruence relation $\equiv$, whose definition is similar to that for SA, and includes the standard rules for changing the orders of parallel compositions and restrictions, and for unfolding recursions.

The side condition of rule PAR-PROC ensures that all subambients of an ambients are activated as soon as possible, before any local reduction takes place (here we exploit the fact that recursions are guarded, otherwise there could be an infinite number of ambients to create).

Local reductions

$$\langle M\rangle \mid (x).\, P \xrightarrow[-:-]{-} P\{^{M}/_{x}\} \gg \mathbf{0} \qquad \text{[LOCAL-COM]}$$

$$\{\mathtt{in}\, n, h\} \mid \{\overline{\mathtt{in}}\, n, k\} \xrightarrow[-:-]{-} \mathbf{0} \gg h\{\mathtt{go}\, k\} \mid k\{\mathtt{OK}\overline{\mathtt{in}}\} \qquad \text{[LOCAL-IN]}$$

$$\{\mathtt{out}\, n, h\} \mid \overline{\mathtt{out}}\, n.\, P \xrightarrow[-:n]{k} P \gg h\{\mathtt{go}\, k\} \qquad \text{[LOCAL-OUT]}$$

$$\mathtt{open}\, n.\, P \mid \{\overline{\mathtt{open}}\, n, h\} \xrightarrow[-:-]{-} \mathtt{wait}.\, P \gg h\{\mathtt{migrate}\} \qquad \text{[LOCAL-OPEN]}$$

Creation

$$h{:}\, n[\, m[\, P\,] \mid Q\,]_{h'} \longmapsto h{:}\, n[\, Q\,]_{h'} \parallel \boldsymbol{\nu} k\, (k{:}\, m[\, P\,]_h) \qquad \text{[NEW-LOCAMB]}$$

$$h{:}\, n[\, \boldsymbol{\nu} m\, P\,]_k \longmapsto \boldsymbol{\nu} m\, (h{:}\, n[\, P\,]_k) \qquad \text{[NEW-RES]}$$

Forwarder

$$h \rhd k \parallel h\{MsgBody\} \longmapsto h \rhd k \parallel k\{MsgBody\} \qquad \text{[FW-MSG]}$$

Consumption of request messages

$$h{:}\, n[\, P\,]_{h'} \parallel h\{Request\} \longmapsto h{:}\, n[\, P \mid \{Request\}\,]_{h'} \qquad \text{[CONSUME-REQ]}$$

Emission of request messages (should be $h \neq \mathtt{root}$)

$$\mathtt{in}\, m.\, P \xrightarrow[h:-]{k} \mathtt{wait}.\, P \gg k\{\mathtt{in}\, m, h\} \qquad \text{[REQ-IN]}$$

$$\overline{\mathtt{in}}\, n.\, P \xrightarrow[h:n]{k} \mathtt{wait}.\, P \gg k\{\overline{\mathtt{in}}\, n, h\} \qquad \text{[REQ-COIN]}$$

$$\mathtt{out}\, m.\, P \xrightarrow[h:-]{k} \mathtt{wait}.\, P \gg k\{\mathtt{out}\, m, h\} \qquad \text{[REQ-OUT]}$$

$$\overline{\texttt{open}}\, n.\, P \xrightarrow[h:n]{k} \texttt{wait}.\, P \gg k\{\overline{\texttt{open}}\, n, h\} \qquad \text{[REQ-COOPEN]}$$

Consumption of completion messages

$$h{:}\, n[\, P \mid \texttt{wait}.\, Q\,]_k \parallel h\{\texttt{go}\, h'\} \longmapsto h{:}\, n[\, P \mid Q\,]_{h'} \qquad \text{[COMPL-PARENT]}$$

$$h{:}\, n[\, P \mid \texttt{wait}.\, Q\,]_k \parallel h\{\overline{\texttt{OKin}}\} \longmapsto h{:}\, n[\, P \mid Q\,]_k \qquad \text{[COMPL-COIN]}$$

$$h{:}\, n[\, P \mid \texttt{wait}.\, Q\,]_k \parallel h\{\texttt{migrate}\} \longmapsto h \rhd k \parallel k\{\texttt{register}\, P \mid Q\} \qquad \text{[COMPL-MIGR]}$$

$$h{:}\, n[\, P \mid \texttt{wait}.\, Q\,]_k \parallel h\{\texttt{register}\, R\} \longmapsto h{:}\, n[\, P \mid Q \mid R\,]_k \qquad \text{[COMPL-REG]}$$

Inference rules

$$\frac{P \xrightarrow[h:n]{k} P' \gg \widetilde{Msg} \qquad Q \text{ does not have unguarded ambients}}{P \mid Q \xrightarrow[h:n]{k} P' \mid Q \gg \widetilde{Msg}} \qquad \text{[PAR-PROC]}$$

$$\frac{P \xrightarrow[h:n]{k} P' \gg \widetilde{Msg}}{h{:}\, n[\, P\,]_k \longmapsto h{:}\, n[\, P'\,]_k \parallel \widetilde{Msg}} \qquad \text{[PROC-AGENT]} \qquad \frac{A \longmapsto A'}{A \parallel B \longmapsto A' \parallel B} \qquad \text{[PAR-AGENT]}$$

$$\frac{A \longmapsto A'}{\boldsymbol{\nu} p\, A \longmapsto \boldsymbol{\nu} p\, A'} \qquad \text{[RES-AGENT]} \qquad \frac{A \equiv A' \qquad A' \longmapsto A'' \qquad A'' \equiv A'''}{A \longmapsto A'''} \qquad \text{[STRUCT-CONG]}$$

5 Correctness of the Abstract Machine

For lack of space we only report the main correctness result. We refer to the full version of the paper, or to the version on the authors's Web page, for more details.

Let $[\![.]\!]$ be the translation of terms of SA into terms of PAN, so defined:

$$[\![P]\!] \stackrel{\text{def}}{=} \texttt{root}{:}\, \texttt{rootname}[\, P\,]_{\texttt{rootparent}}$$

We write $A \Downarrow_n$ if A is *observable at* n; this means, intuitively, that A contains an agent n that accepts interactions with the external environment. Formally: $A \downarrow_n$ if $A \equiv \boldsymbol{\nu} \widetilde{p}\, (h{:}\, n[\, \mu.\, Q_1 \mid Q_2\,]_{\texttt{root}} \parallel A')$ where $\mu \in \{\overline{\texttt{in}}\, n, \overline{\texttt{open}}\, n\}$ and $n \notin \widetilde{p}$. Then, using $\Longmapsto$ for the reflexive and transitive closure of $\longmapsto$, we write $A \Downarrow_n$ if $A \Longmapsto \downarrow_n$. Observability in SA is defined similarly: $P \Downarrow_n$ if $P \Longrightarrow P'$, for some P' such that $P' \equiv \boldsymbol{\nu} \widetilde{n}\, (n[\, \mu.\, Q_1 \mid Q_2\,] \mid Q_3)$ where $\mu \in \{\overline{\texttt{in}}\, n, \overline{\texttt{open}}\, n\}$ and $n \notin \widetilde{n}$.

Theorem 1. *Let $P \in SA$. It holds that $P \Downarrow_n$ iff $[\![P]\!] \Downarrow_n$, for all n.*

The key steps in the proof of Theorem 1 are the following. First, since PAN separates between the logical and physical distribution of ambients, we need to make sure that the two are consistent. For instance, the graph of the dependencies among locations in the physical distribution, which represents the logical structure, should be a tree. We also need conditions that ensure that the `wait` state is used as described informally in previous sections. We therefore introduce the notion of *well-formed nets* and then prove that well-formedness is invariant under reductions. Secondly, we prove that on well-formed nets administrative reductions do not affect behavioural equivalences, where a reduction $A \longmapsto A'$ is *administrative* if its derivation proof does not use the axioms of local reductions. Thirdly, we establish an operational correspondence between the reductions of a well-typed SA process and of its PAN translation.

6 Immobile Ambients

The other important type of ambients in SA are the *immobile* ambients. (A typed SA program may therefore contain both single-threaded *and* immobile ambients.) These are ambients that: (i) cannot jump into or out of other ambients; (ii) cannot be opened. Thus the only capabilities that an immobile ambient can execise are $\overline{\texttt{in}}\,n$, $\overline{\texttt{out}}\,n$, and $\texttt{open}\,n$; several of them can be ready for execution at the same time.

The same rules for the abstract machine in Section 4 could be adopted for immobile ambients. This has however the following problem. Consider the process

$$P \stackrel{\text{def}}{=} n[\,\texttt{rec}\,X.\,(\overline{\texttt{in}}\,n \mid \boldsymbol{\nu} m\,(\texttt{open}\,m.\,X \mid m[\,\overline{\texttt{open}}\,m\,]))\,]$$

(Using replication, the behaviour of P can be expressed as $n[\,!\overline{\texttt{in}}\,n\,]$.) With the rules of Section 4, ambient n could flood its parent with $\overline{\texttt{in}}$ requests. To avoid the problem, we modify PAR-PROC:

$$\frac{\begin{array}{l}n \text{ is an immobile ambient}\\ Q \text{ does not have unguarder ambients}\\ P \xrightarrow[h:n]{k} P' \gg \widetilde{Msg} \quad Q \text{ or } P' \text{ do not contain any } \texttt{wait}\end{array}}{P \mid Q \xrightarrow[h:n]{k} P' \mid Q \gg \widetilde{Msg}} \qquad \text{[IMM-PAR-PROC]}$$

We then have to modify also LOCAL-OPEN and PAR-PROC, so that an immobile ambient does not go into a `wait` state while opening a child ambient:

$$\frac{n \text{ is an immobile ambient}}{\texttt{open}\,m.\,P \mid \{\overline{\texttt{open}}\,m, h\} \xrightarrow[-:n]{-} P \gg h\{\texttt{migrate}\}} \qquad \text{[IMM-LOCAL-OPEN]}$$

$$\frac{n \text{ is an immobile ambient}}{h{:}\,n[\,P\,]_k \parallel h\{\texttt{register}\,R\} \longmapsto h{:}\,n[\,P \mid R\,]_k} \qquad \text{[IMM-COMPL-REG]}$$

The original rules LOCAL-OPEN, PAR-PROC, and COMPL-REG are now used only for ST ambients, therefore the corresponding side conditions is added.

With the new rules, the following property holds (for both ST and immobile ambients): an agent can send only one request message at a time to its parent. An immobile ambient can exercise several capabilities at the same time. Sending one request at a time to the parent is correct because the only capability that may produce a request from an immobile ambient named n to its parent is $\overline{\texttt{in}}\,n$ (the protocol for $\overline{\texttt{in}}$ can however be executed in parallel with several protocols for $\overline{\texttt{out}}$ and `open` operations). With the new rules, the addition of immobile ambients requires few modifications to the correctness proof of Section 5.

7 Comparisons and Remarks

Cardelli [2,3] has produced the first implementation, called *Ambit*, of an ambient-like language; it is a single-machine implementation of the untyped Ambient calculus, and is written in Java. The algorithms are based on locks: all the ambients involved in a movement (three ambients for an `in` or `out` movement,

two for an `open`) have to be locked for the movement to take place. More recently, Fournet, Lévy and Schmitt [9] have presented a distributed implementation of the untyped Ambient calculus, as a translation of the calculus into Jocaml [10] (a programming language based on the distributed Join Calculus [8]). Our abstract machine is quite different from the above mentioned implementations mainly because:

(i) We are implementing a variant of the Ambient calculus (the Safe Ambients) that has coactions and types for single-threadness and immobility.
(ii) We separate the logical and physical distribution of an ambient system.

The combination of (i) and (ii) allows us considerable simplifications, both in the abstract machine and in its correctness proof. We are not aware of correctness proofs for Ambit. The correctness proof for the Join implementation is very ingenious and makes use of sophisticated techniques, such as coupled simulation and decreasing diagram techniques. Below, we focus on the differences with the Join implementation, which is a distributed implementation, and which we will refer to as AtJ (Ambients to Join).

- In AtJ `open` is by far the most complex operation, because the underlying Jocaml language does not have primitives with a similar effect. In AtJ, every ambient has a manager that collects the requests of operations from the sub-ambients and from the local processes. If the ambient is opened, its manager becomes a forwarder of messages towards the parent ambient. The processes local to the opened ambient are not moved.
  As a consequence, in AtJ the processes local to an ambient can be distributed on several locations. Therefore, also the implementation of the communication rule R-MSG may require exchange of messages among sites, which does not occur in PAN, where forwarders are always empty.
- In AtJ, forwarders are also introduced with `in` and `out` operations, to cope with possible asynchronous messages still travelling after the move is finished. These forwarders are not needed in PAN.
- In PAN, the presence of coactions dispenses us from having backward pointers from an ambient to its children. In the example of the first row of Figure 1, without $\overline{\texttt{in}}$, ambient c would not know the location of b and therefore could not communicate this location to a. Backward pointers, as in AtJ, make bookkeeping and correctness proof more complex.
  In PAN, the absence of backward pointers and the presence of coactions make the implementation of forms of dynamic linking straightforward: new machines hosting ambients can be connected to existing machine running an ambient system; it suffices that the new machines know the location of one of the running ambients; no modifications or notifications is needed to the running ambients themselves.
- In PAN, since any moving ambient (an ambient that tries to enter or exit another ambient, or that can be opened) is single-threaded, each ambient requests at most one operation at a time to its parent. By contrast, in AtJ an ambient can send an unbounded number of requests to the parent (an example is $n[\,!\texttt{in}\,m_1 \mid !\texttt{out}\,m_2\,]$).

Moreover, due to this property, in PAN no ambient needs a log of pending requests received from a given children or sent to the parent. Without the property, both forms of log are needed, as it happens in AtJ. To see why, consider two ambients a and b, where b is the parent of a. If moving ambients can request several operations concurrently, b must of course keep a log of the pending requests from a. A copy of the same log must however be kept by a, because messages exchanged among ambients are asynchronous and therefore the following situation could arise. Suppose a requests two operations, say $\texttt{in}\, n$ and $\texttt{in}\, m$. The request for $\texttt{in}\, n$ could reach b first. The request for $\texttt{in}\, m$ could reach b only when the movement for $\texttt{in}\, n$ has been completed (indeed, a might have completed other movements). The request $\texttt{in}\, m$ must now be resent to the new parent of a, but b does not possess this information. This task must therefore be accomplished by a, which, for this, must have stored $\texttt{in}\, m$ in its log of pending requests to the parent.
The example also shows that, aside from message retransmission in forwarders, some requests may have to be retransmitted several times, to different parents (in the example, $\texttt{in}\, m$); in PAN every request is sent at most once.

- In PAN, any movement for a given ambient is requested to the parent, which (assuming this is not a forwarder) makes decisions and gives authorisations; the grandparent is never contacted. This homogeneity property breaks in presence of backward pointers from an ambient to its children. For instance, the simulation of the `out` reduction in the second row of Figure 1 would then need also the involvement of the grandparent c.
- In AtJ, the domain of physical distribution is a tree. The `in` and `out` operations produce physical movements in which an ambient, and all its tree of subambients, must move. To achieve this, the tree of ambients is first "frozen" so that all the activities in the ambients of the tree stop while the movement takes place. In PAN, where the domain of physical distribution is flat, `in` and `out` only give logical movement; no freezing of ambients is required. On the other hand, in PAN, but not in AtJ, `open` gives physical movement.
- PAN is an abstract machine, and is therefore independent of a specific target language.

8 Implementation Architecture

Our implementation, written in Java, follows the definition of the abstract machine (as usual in process calculi, rules for arbitrary changing the order of parallel components need some randomisation mechanism to be implemented; we do not do this, which may reduce non-determinism). Perhaps the main difference is that the implementation allows clustering of agents on the same IP node (i.e. a physical machine). Therefore the implementation is made of three layers: agents, nodes and the network. The address k of an agent is composed of the *IP-name* of the node on which its resides, plus a suffix, which is different for each agent in that node. Each agent is executed by an independent Java thread; the processes

local to an ambient are scheduled using a round-robin policy. Each agent knows its name, its address, its parent's address, and keeps a link to its node.

From a physical point of view, the messages exchanged between agents are of two kinds: local, when both agents reside on the same node, and remote, when two distinct nodes are involved. In each node a special Java *RMI object*, with its own thread of execution, takes care of inter-node communications. For this, nodes act alternatively as clients (requiring that a message is sent to another computer) and as servers (receiving a message and pushing it into a local mailbox). The node layer is implemented using Java RMI and *serialization*, and the network layer simply provides IP-name registry for RMI communications to take place (using Java *RMIregistry*).

An agent acts as an interpreter for the ambient expressions that constitute its local processes. When the agent wants to create a subambient, it sends a special message to its node, which will spawn a new agent hosting the subambient code. We also allow *remote creation* of new agents: an agent may send a message to a node different from its own, to demand the creation of a subambients. This corresponds to the addition of a primitive $\texttt{create}\ n[\,P\,]\ \texttt{at}\ h$, where h is the IP-name of a node, to the abstract machine. When the execution of an ambient expression begins on a given node, the first action is the local creation of a *root* agent. An agent resides on the same node until it is opened; then, its processes are serialised and sent via RMI to the parent agent. The implementation also allows dynamic linking of ambients, as hinted at in Section 7.

9 Further Developments

In the abstract machine presented, a message may have to go through a chain of forwarders before getting to destination. A (partial) solution to this problem is a modification of the rules that guarantees the following property: every agent sends a message to a given forwarder at most once. The modification consists in adding the source field to the completion messages $h\{\texttt{OK}\overline{\texttt{in}}\}$, which thus becomes $h\{\texttt{OK}\overline{\texttt{in}}, k\}$, where k is the ambient that is authorising the move. Thus the rules LOCAL-IN and COMPL-COIN become

$$\{\texttt{in}\ n, h\} \mid \{\overline{\texttt{in}}\ n, k\} \xrightarrow[h':-]{-} \mathbf{0} \gg h\{\texttt{go}\ k\} \parallel k\{\texttt{OK}\overline{\texttt{in}}, h'\} \qquad \text{[LOCAL-IN2]}$$

$$h{:}\, n[\,P \mid \texttt{wait}.\,Q\,]_k \parallel h\{\texttt{OK}\overline{\texttt{in}}, h'\} \longmapsto h{:}\, n[\,P \mid Q\,]_{h'} \qquad \text{[COMPL-COIN2]}$$

The reason why these rules may be useful is that the parent of an ambient that has sent a $\overline{\texttt{in}}$ request may have become a forwarder; thus the real parent is another ambient further up in the hierarchy. With the new rules, the parent of the ambient that has sent the $\overline{\texttt{in}}$ request is updated and hence this ambient will not go through the forwarder afterwards. With the other capabilities that may originate a request from an ambient to is parent (`open`, `out`, `in`), the issue does not arise, because either the requesting ambient is dissolved (`open`), or its parent is anyway modified (`out`, `in`).

Even with the rules above, however, the forwarder introduced in an `open` operation is permanent. We plan to study the problem of the garbage-collection

of forwarders. We also plan to experiment the addition of backwards pointers, from an ambient to its children; this should avoid the introduction of forwarders in an `open`, but may complicate other parts of the abstract machine.

In the abstract machine, `open` is the only operation that gives movement of terms. Although at present we do not see the need of enhancing this, the modifications for allowing movement of terms also with `in` and `out` would be simple. The main price is the introduction of additional forwarders, as we have now in the `open` case.

Acknowledgements. We have benefitted from comments by Jean-Jacques Lévy and Alan Schmitt.

References

1. M. Bugliesi, G. Castagna. Secure safe ambients. *28th POPL*. 2001.
2. L. Cardelli. Ambit. `http://www.luca.demon.co.uk/Ambit/Ambit.html` 1997.
3. L. Cardelli. Mobile ambient synchronisation. Technical Report 1997-013, Digital SRC, 1997.
4. L. Cardelli, A.D. Gordon. Mobile ambients. *FoSSaCS '98*, LNCS 1378, 1998.
5. L. Cardelli, A.D. Gordon. Equational properties of mobile ambients. *FoSSaCS'99*, LNCS 1578, 1999.
6. L. Cardelli, A.D. Gordon. Types for mobile ambients. *26th POPL*, 1999.
7. L. Cardelli, A.D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. *27th POPL*, 2000.
8. C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, D. Rémy. A calculus of mobile processes. *CONCUR '96*, LNCS 1119, 1996.
9. C. Fournet, J.-J. Lévy, A. Schmitt. An asynchronous distributed implementation fo mobile ambients. *IFIP TCS2000*, LNCS 1872, 2000.
10. F. Le Fessant. The Jocaml system prototype. `join.inria.fr/jocaml`. 1998.
11. F. Levi, D. Sangiorgi. Controlling interference in ambients. *27th POPL*, 2000.
12. F. Nielson, H.R. Nielson Shape analysis for mobile ambients. *27th POPL*, 2000.

Towards Quantitative Verification of Probabilistic Transition Systems

Franck van Breugel[1] and James Worrell[2]

[1] York University, Department of Computer Science
4700 Keele Street, Toronto, M3J 1P3, Canada
[2] Tulane University, Department of Mathematics
6823 St Charles Avenue, New Orleans LA 70118, USA

Abstract. It has been argued that Boolean-valued logics and associated discrete notions of behavioural equivalence sit uneasily with semantic models featuring quantitative data, like probabilistic transition systems. In this paper we present a pseudometric on a class of reactive probabilistic transition systems yielding a quantitative notion of behavioural equivalence. The pseudometric is defined via the terminal coalgebra of a functor based on the Hutchinson metric on the space of Borel probability measures on a metric space. We also characterize the distance between systems in terms of a real-valued modal logic.

1 Introduction

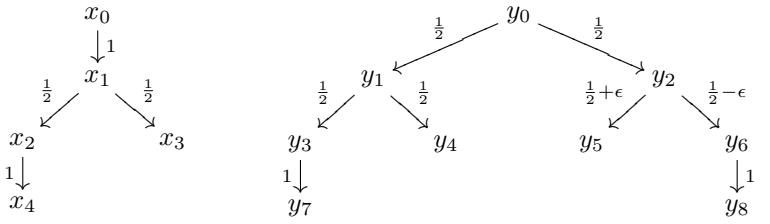
The majority of verification methods for concurrent systems only produce qualitative information. Questions like "Does the system satisfy its specification?" and "Do the systems behave the same?" are answered "Yes" or "No". Huth and Kwiatkowska [13] and Desharnais, Gupta, Jagadeesan and Panangaden [8] have pointed out that such discrete Boolean-valued reasoning sits uneasily with continuous semantic models like probabilistic transition systems. For instance, the probabilistic modal logic of Larsen and Skou [16] adds probability thresholds to traditional modal logic. In this logic one has a formula like $\diamond_q \phi$ which is satisfied if the sum of the probabilities of transitions to states satisfying ϕ exceeds $q \in [0, 1]$. Such a formalism does not support approximate reasoning: any inexactness in the calculation of the semantics of ϕ may result in an incorrect conclusion as to the truth or falsity of $\diamond_q \phi$. This is particularly problematic if one wants to reason about infinite state systems in terms of finite approximants.

In a similar vein, Desharnais et al. [7] and Giacalone, Jou and Smolka [10] have criticized all-or-nothing notions of operational equivalence for probabilistic systems such as Larsen and Skou's probabilistic bisimulation [16]. Recall that a probabilistic bisimulation is an equivalence relation on the state space of a

[1] Supported by Natural Sciences and Engineering Research Council of Canada.
[2] Supported by the US Office of Naval Research.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 421–432, 2001.

transition system such that related states have exactly the same probability of making a transition into any equivalence class. Thus, for instance, the probabilistic transition systems



are only probabilistic bisimilar if ϵ is 0. However, the two systems behave almost the same for very small ϵ different from 0. In the words of [7] behavioural equivalences like probabilistic bisimilarity are not robust, since they are too sensitive to the exact probabilities of the various transitions.

To address some of the issues raised above, Huth and Kwiatkowska introduce a non-standard semantics for formulas of the modal μ-calculus over a probabilistic transition system. Formulas take truth values in the unit interval $[0, 1]$: in particular the modal connective is interpreted by integration. A related though quite distinct real-valued modal logic is introduced by Desharnais et al. [7]. Their logic is used to define a notion of approximate equivalence for probabilistic transition systems; this is formalized as a pseudometric[1] on the class of all such systems. The pseudometric is intended to provide for compositional reasoning about the approximate equivalence of concurrent interacting probabilistic systems. Processes are at 0 distance just in case they are probabilistic bisimilar.

Many different kinds of transition system can be viewed as coalgebras; Rutten [19] provides numerous examples. De Vink and Rutten [21] have shown that both discrete and continuous (labelled) probabilistic transition systems can be seen as coalgebras. By viewing these systems as coalgebras one can transfer results from the theory of coalgebra to the setting of probabilistic systems. This theory includes a general definition of bisimilarity which De Vink and Rutten studied for probabilistic transition systems with discrete and ultrametric state spaces.

In this paper we obtain a metric-space domain for reactive probabilistic processes as the terminal coalgebra of an endofunctor F on the category of pseudometric spaces and nonexpansive maps. The definition of F is based on the Hutchinson metric on probability measures [14]. F-coalgebras can be seen as reactive probabilistic transition systems with discrete or continuous state spaces. Unlike the terminal coalgebras studied by De Vink and Rutten [21] and Baier and Kwiatkowska [3] the metric on our domain varies continuously with transition probabilities. It provides for a notion of approximate equivalence of probabilistic processes similar to the pseudometric of Desharnais et al. mentioned above. In fact, we define a pseudometric on the state space of a reactive transition system (seen as an F-coalgebra) as the metric kernel of the unique map to the terminal

[1] A pseudometric differs from an ordinary metric in that different elements can have distance 0.

F-coalgebra. That is, the distance between two states is the distance between their images under the unique map to the terminal coalgebra. We show that our pseudometric can also be obtained by adding negation to the logic of Desharnais et al. Furthermore we compare our pseudometric with the distance functions of De Vink and Rutten and of Norman.

2 The Pseudometric

In this section we introduce an endofunctor on the category of pseudometric spaces and nonexpansive maps based on the Hutchinson metric. We prove that the functor has a terminal coalgebra, and we use this to define our pseudometric on reactive probabilistic transition systems.

In [14], Hutchinson introduced a metric on the set of Borel probability measures on a metric space. Here, we generalize his definition to pseudometric spaces. We restrict ourselves to spaces whose points have distance at most 1, since they serve our purpose and simplify the definition of the distance function a little. Let X be a 1-bounded pseudometric space. We denote the set of Borel probability measures on X by $\mathcal{M}(X)$.

Definition 1. *The* Hutchinson metric[2] $d_{\mathcal{M}(X)} : \mathcal{M}(X) \times \mathcal{M}(X) \to [0,1]$ *is defined by*

$$d_{\mathcal{M}(X)}(\mu_1, \mu_2) = \sup \left\{ \int_X f d\mu_1 - \int_X f d\mu_2 \mid f \in X \to [0,1] \text{ is nonexpansive} \right\}.$$

A function is nonexpansive if it does not increase any distances. For a proof that $d_{\mathcal{M}(X)}$ is a 1-bounded pseudometric we refer the reader to, for example, Edgar's textbook [9, Proposition 2.5.14].

In a pseudometric space, compactness is a natural generalization of finiteness. In the rest of this paper, we focus on Borel probability measures which are completely determined by their values for the compact subsets of the space X.

Definition 2. *A Borel probability measure μ on X is* tight *if for all $\epsilon > 0$ there exists a compact subset K_ϵ of X such that $\mu(X \setminus K_\epsilon) < \epsilon$.*

Under quite mild conditions on the space, for example, completeness and separability, every measure is tight (see, for example, Parthasarathy's textbook [18, Theorem II.3.2]). We denote the set of tight Borel probability measures on X by $\mathcal{M}_t(X)$. We are interested in these tight measures because of the following

Theorem 1.

1. *X is complete if and only if $\mathcal{M}_t(X)$ is complete.*
2. *X is compact if and only if $\mathcal{M}_t(X)$ is compact.*

Proof. See, for example, [9, Theorem 2.5.25]. □

[2] The Hutchinson metric is also known as the Kantorovich metric.

$\mathcal{M}_t$ can be extended to an endofunctor on the category $\mathcal{PM}et_1$ of 1-bounded pseudometric spaces and nonexpansive functions as follows. Let X and Y be 1-bounded pseudometric spaces. Let $f : X \to Y$ be a nonexpansive function.

Definition 3. *The function* $\mathcal{M}_t(f) : \mathcal{M}_t(X) \to \mathcal{M}_t(Y)$ *is defined by*

$$\mathcal{M}_t(f)(\mu) = \mu \circ f^{-1}.$$

It is readily verified that the measure $\mathcal{M}_t(f)(\mu)$ is tight, that the function $\mathcal{M}_t(f)$ is nonexpansive and that the action of $\mathcal{M}_t$ on arrows is functorial.

Next we state and explain a property of $\mathcal{M}_t$ which will later allow us to exploit the terminal coalgebra theorem of Turi and Rutten [20].

Proposition 1. *The functor* $\mathcal{M}_t$ *is locally nonexpansive: for all nonexpansive functions* $f_1,\ f_2 \in X \to Y$*,*

$$d_{\mathcal{M}_t(X) \to \mathcal{M}_t(Y)}(\mathcal{M}_t(f_1), \mathcal{M}_t(f_2)) \leq d_{X \to Y}(f_1, f_2).$$

□

A continuous probabilistic transition system with label set L consists of a 1-bounded pseudometric space X of states together with a Borel subprobability measure $\mu_{l,x}$ for each label l and state x. The transition function μ_l is a conditional probability determining the reaction of the system to an action l selected by the environment. $\mu_{l,x}$ assigns to each Borel set $B \subseteq X$ the probability that the system makes a transition to a state in the set B given that it was in the state x before the action l. We consider Borel subprobability measures[3] to allow for the possibility that the system may refuse l. We also require that for each Borel set B the map $x \mapsto \mu_{l,x}(B)$ is measurable, i.e. that $\lambda x \lambda B.\mu_{l,x}(B)$ is a stochastic kernel. This is the so-called reactive model of probabilistic systems. For a detailed discussion of the importance of studying these continuous systems, rather than concentrating on the discrete ones, we refer the reader to the work of Desharnais et al. [5,6]. In the present paper we stick to discrete systems when we come to exemplify our work. Also, for ease of exposition, we only consider unlabelled transition systems. That is, we assume L is a singleton space and write μ_x for $\mu_{l,x}$. Our results extend easily to the labelled case.

A discrete probabilistic transition system is just a special case of a continuous one where the metric on the state space is discrete and the transition probability is given by a subprobability distribution. We can picture such a system as a directed graph with arcs labelled by probabilities: there is no need to mention Borel sets.

Next, we demonstrate that a large class of continuous probabilistic transition systems can be viewed as coalgebras. But first we review some basic notions.

Definition 4. *Let* $\mathcal{C}$ *be a category. Let* $F : \mathcal{C} \to \mathcal{C}$ *be a functor. An* F-coalgebra *consists of an object* C *in* $\mathcal{C}$ *together with an arrow* $f : C \to F(C)$ *in* $\mathcal{C}$*. An*

[3] In a *sub*probability measure $\mu_{l,x}$ we have that $\mu_{l,x}(X) \leq 1$ rather than $\mu_{l,x}(X) = 1$.

F-homomorphism from F-coalgebra $\langle C, f\rangle$ to F-coalgebra $\langle D, g\rangle$ is an arrow $h : C \to D$ in $\mathcal{C}$ such that $F(h) \circ f = g \circ h$.

$$\begin{array}{ccc} C & \xrightarrow{\quad h \quad} & D \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ F(C) & \xrightarrow[F(h)]{} & F(D) \end{array}$$

The F-coalgebras and F-homomorphisms form a category. The terminal object in this category, if it exists, is called the terminal F-coalgebra.

We consider the functor

$$F = \tfrac{1}{2} \cdot \mathcal{M}_t(\mathbf{1} + -) : \mathcal{PM}et_1 \to \mathcal{PM}et_1,$$

where $\mathbf{1}$ is the terminal object[4] functor, $+$ is the coproduct[5] functor, $\mathcal{M}_t$ is the Hutchinson functor introduced above, and $\frac{1}{2}\cdot$ is the scaling[6] functor. An F-coalgebra consists of a 1-bounded pseudometric space X together with a nonexpansive function $\mu : X \to \frac{1}{2} \cdot \mathcal{M}_t(\mathbf{1} + X)$. A continuous probabilistic transition system such that

- for all states x, the Borel probability measure μ_x is tight, and
- for all states x_1, x_2, $\frac{1}{2} \cdot d_{\mathcal{M}_t(1+X)}(\mu_{x_1}, \mu_{x_2}) \leq d_X(x_1, x_2)$,

can be viewed as an F-coalgebra. For now we observe that this class certainly includes all discrete probabilistic transition systems, and we refer the reader forward to the conclusion for further discussion of these two restrictions.

Theorem 2. *There exists a terminal F-coalgebra.*

Proof. Since the functors $\mathbf{1}$, $+$, and $\mathcal{M}_t$ are locally nonexpansive (Proposition 1) and the scaling functor $\frac{1}{2}\cdot$ is locally contractive, the functor F is locally contractive. According to Theorem 1, the functor $\mathcal{M}_t$, and hence the functor F, preserves the subcategory $\mathcal{CM}et_1$ of 1-bounded complete metric spaces and nonexpansive functions. According to [20, Theorem 7.3], this functor restricted to $\mathcal{CM}et_1$ has a terminal coalgebra $\langle fix(F), \iota\rangle$. It is not too hard to see from the proof of that theorem that $\langle fix(F), \iota\rangle$ is also a terminal F-coalgebra. □

[4] The terminal object of $\mathcal{PM}et_1$ is the singleton space.

[5] The coproduct object of the objects X and Y in $\mathcal{PM}et_1$ is the disjoint union of the sets underlying the spaces X and Y endowed with the pseudometric

$$d_{X+Y}(v, w) = \begin{cases} d_X(v, w) & \text{if } v \in X \text{ and } w \in X \\ d_Y(v, w) & \text{if } v \in Y \text{ and } w \in Y \\ 1 & \text{otherwise.} \end{cases}$$

[6] The scaling by $\frac{1}{2}\cdot$ of an object in $\mathcal{PM}et_1$ leaves the set unchanged and multiplies all distances by a half.

The distance in the terminal coalgebra is a trade-off between the depth of observations needed to distinguish systems, and the amount each observation differentiates the systems. The relative weight given to these two factors is determined by the contraction introduced in the definition of the functor F.

Now we present the definition of our pseudometric on probabilistic transition systems. Instead of directly defining a pseudometric on systems, we define it on the states of a system. The distance between two systems can be obtained by combining the two systems into one and taking the distance between the initial states of the original systems in the combined one. For a continuous probabilistic system represented by the F-coalgebra $\langle X, \mu\rangle$, let us write $[\![-]\!]_{\langle X,\mu\rangle}$ for the unique map $\langle X, \mu\rangle \to \langle \mathit{fix}\,(F), \iota\rangle$.

Definition 5. *The distance function* $d_{\mathrm{H}} : X \times X \to [0,1]$ *is defined by*

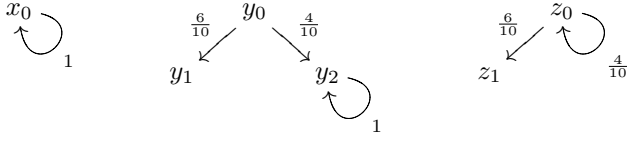
$$d_{\mathrm{H}}\,(x_1, x_2) = d_{\mathit{fix}\,(F)}\,([\![x_1]\!]_{\langle X,\mu\rangle}, [\![x_2]\!]_{\langle X,\mu\rangle})$$

Note that we now have two pseudometrics on the state space X: the original pseudometric d_X which defines the Borel sets and the above introduced pseudometric d_{H} which captures the difference in behaviour in a quantitative way. Since the function $[\![-]\!]_{\langle X,\mu\rangle}$ is nonexpansive, the d_X-distances are greater than or equal to the d_{H}-distances.

3 Desharnais, Gupta, Jagadeesan, and Panangaden

We compare our pseudometric with the one introduced by Desharnais, Gupta, Jagadeesan and Panangaden in [7]. We argue that our distances are more intuitive than theirs. Furthermore, we extend one of their definitions a little and show that the pseudometric so obtained coincides with ours.

Consider the following three probabilistic transition systems.



The first system terminates with probability 0, the second one with probability $\frac{6}{10}$ and the third one with probability 1. The probability that the systems make, for example, at most three transitions is 0, $\frac{6}{10}$ and $\frac{113}{125}$, respectively. Based on these kind of observations, one may infer that the first system behaves more like the second one than the third one. This is reflected by our pseudometric, since the first and second system are $\frac{3}{20}$ apart whereas the first and third system are at distance $\frac{3}{16}$. However, in the pseudometric introduced by Desharnais et al. both the first and the second system and the first and third system are $\frac{3}{20}$ apart.

Desharnais et al. defined their pseudometric in terms of a real-valued logic. Their work builds on an idea of Kozen [15] to generalize logic to handle probabilistic phenomena. An extension of their real-valued modal logic is introduced in the following definition.

Definition 6. *The set $\mathcal{F}$ of* functional expressions *is defined by*

$$f ::= 1 \mid \diamond f \mid \max(f, f) \mid 1 - f \mid f \dot{-} q$$

where q is a rational in $[0,1]$.

Informally, there is the following correspondence between functional expressions and formulae in the probabilistic modal logic of Larsen and Skou [16] (see also [5, 6]). True is represented by 1, disjunction is represented by max, negation by $1-$, and the connective $\diamond_q$ decomposes as $\diamond$ and $\dot{-}\, q$. The main difference between the above definition of functional expressions and the one presented by Desharnais et al. is the presence of negation.[7]

Given a continuous probabilistic transition system represented by the F-coalgebra $\langle X, \mu\rangle$, each functional expression f can be interpreted as a function $f_{\langle X,\mu\rangle}$ from X to $[0,1]$ as follows.

Definition 7. *For each $f \in \mathcal{F}$, the function $f_{\langle X,\mu\rangle} : X \to [0,1]$ is defined by*

$$\begin{aligned} 1_{\langle X,\mu\rangle}(x) &= 1 \\ (\diamond f)_{\langle X,\mu\rangle}(x) &= \tfrac{1}{2} \cdot \textstyle\int_X f_{\langle X,\mu\rangle}\, d\mu_x \\ (\max(f,g))_{\langle X,\mu\rangle}(x) &= \max\left(f_{\langle X,\mu\rangle}(x), g_{\langle X,\mu\rangle}(x)\right) \\ (1-f)_{\langle X,\mu\rangle}(x) &= 1 - f_{\langle X,\mu\rangle}(x) \\ (f \dot{-} q)_{\langle X,\mu\rangle}(x) &= f_{\langle X,\mu\rangle}(x) \dot{-} q \end{aligned}$$

where

$$r \dot{-} q = \begin{cases} r - q & \text{if } r \geq q \\ 0 & \text{otherwise.} \end{cases}$$

It is readily verified that for all $f \in \mathcal{F}$ the function $f_{\langle X,\mu\rangle}$ is nonexpansive, and hence measurable. The functional expressions induce a pseudometric as follows.

Definition 8. *The distance function $d_{\mathrm{DGJP}} : X \times X \to [0,1]$ is defined by*

$$d_{\mathrm{DGJP}}(x_1, x_2) = \sup_{f \in \mathcal{F}} f_{\langle X,\mu\rangle}(x_1) - f_{\langle X,\mu\rangle}(x_2).$$

Clearly, the above introduced distance function is a 1-bounded pseudometric. Now we have three different distance functions on the state space X: d_X, d_{H} and d_{DGJP}. To distinguish these three pseudometric spaces we denote them by $\langle X, d_X\rangle$, $\langle X, d_{\mathrm{H}}\rangle$ and $\langle X, d_{\mathrm{DGJP}}\rangle$. Since the functions $f_{\langle X,\mu\rangle}$ are nonexpansive, the d_X-distances are greater than or equal to the d_{DGJP}-distances.

In the rest of this section, we give an outline of a proof that d_{H} and d_{DGJP} coincide. In fact we concentrate on proving the inequality $d_{\mathrm{H}} \leq d_{\mathrm{DGJP}}$, the converse being more straightforward. To this end we introduce a transition function μ' such that $\langle\langle X, d_{\mathrm{DGJP}}\rangle, \mu'\rangle$ is an F-coalgebra. Since the d_X-distances are greater than or equal to the d_{DGJP}-distances, every Borel set on $\mathbf{1} + \langle X, d_{\mathrm{DGJP}}\rangle$ is a Borel set on $\mathbf{1} + \langle X, d_X\rangle$. Therefore, we can define for every $x \in X$ the Borel probability measure μ'_x as μ_x restricted to the Borel sets on $\mathbf{1} + \langle X, d_{\mathrm{DGJP}}\rangle$. Of course, we have to check that μ' is an arrow in $\mathcal{PM}et_1$.

[7] In a draft version, but not in the final version, of [8] negation was considered.

Proposition 2. *The function μ' is nonexpansive.*

Proof. Let $\epsilon > 0$. Let $x \in X$. Since the measure μ_x is tight, there exists a compact subset K of $\langle X, d_X\rangle$ such that $\mu_x(X \setminus K) < \epsilon$. Since the d_X-distances are greater than or equal to the d_{DGJP}-distances, K is also a compact subset of $\langle X, d_{\mathrm{DGJP}}\rangle$. From the definition of μ' we can conclude that $\mu'_x(X \setminus K) < \epsilon$.

Let $g : X \to [0,1]$ be a function which is nonexpansive with respect to d_{DGJP}. Then there exists a functional expression f such that $g \restriction K$ and $f_{\langle X,\mu\rangle} \restriction K$ are at most ϵ apart. This can be proved by exploiting [1, Lemma A.7.2][8]. Using all the above, we can show that

$$\int_X g\, d\mu'_x \text{ and } \int_X f_{\langle X,\mu\rangle}\, d\mu_x \text{ are at most } 3\epsilon \text{ apart.} \tag{1}$$

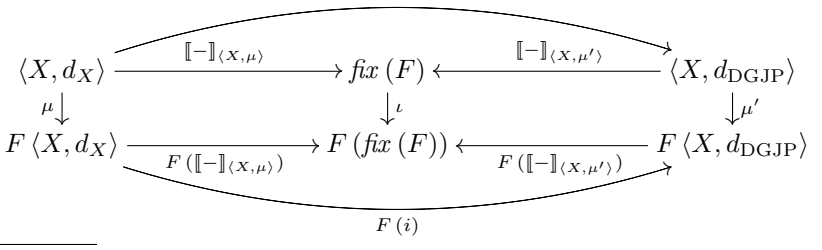
Let $x_1, x_2 \in X$. Without loss of generality, we may assume that $\mu'_{x_1}(\mathbf{1}) \leq \mu'_{x_2}(\mathbf{1})$. Hence,

$$\begin{aligned}
&d_{F\langle X,d_{\mathrm{DGJP}}\rangle}(\mu'_{x_1}, \mu'_{x_2}) \\
&= \tfrac{1}{2} \cdot \sup\left\{ \int_X g\, d\mu'_{x_1} - \int_X g\, d\mu'_{x_2} \mid g \in \langle X, d_{\mathrm{DGJP}}\rangle \to [0,1] \text{ is nonexpansive} \right\} \\
&\leq \tfrac{1}{2} \cdot \sup_{f\in\mathcal{F}} \int_X f_{\langle X,\mu\rangle}\, d\mu_{x_1} - \int_X f_{\langle X,\mu\rangle}\, d\mu_{x_2} \quad [(1)] \\
&= \sup_{f\in\mathcal{F}} (\diamond f)_{\langle X,\mu\rangle}(x_1) - (\diamond f)_{\langle X,\mu\rangle}(x_2) \\
&\leq \sup_{f\in\mathcal{F}} f_{\langle X,\mu\rangle}(x_1) - f_{\langle X,\mu\rangle}(x_2) \\
&= d_{\mathrm{DGJP}}(x_1, x_2),
\end{aligned}$$

that is, μ' is nonexpansive. □

Note that $\diamond$, min and max (which is a combination of min and $1 - $) play a role in the above proof. Also $\dot{-}\, q$ and 1 are needed in some of the details of the proof which are not presented here.

One can easily verify that the nonexpansive function i from $\langle X, d_X\rangle$ to $\langle X, d_{\mathrm{DGJP}}\rangle$ mapping x to x is an F-homomorphism.



[8] Let K be a compact Hausdorff space. Let A be a set of the real-valued continuous functions on K such that $f \in A$ and $g \in A$ implies $\max(f,g) \in A$ and $\min(f,g) \in A$. If a function f can be approximated at each pair of points by functions in A then f is in the closure of A.

Hence, $[\![-]\!]_{\langle X,\mu\rangle}$ and $[\![-]\!]_{\langle X,\mu'\rangle} \circ i$ are both F-homomorphisms from $\langle X, d_X\rangle$ to $\mathit{fix}\,(F)$. Since $\mathit{fix}\,(F)$ is terminal they are equal, i.e. for all $x \in X$,

$$[\![x]\!]_{\langle X,\mu\rangle} = [\![x]\!]_{\langle X,\mu'\rangle} \tag{2}$$

Theorem 3. *For all* $x_1,\ x_2 \in X,\ d_{\mathrm{H}}\,(x_1, x_2) \leq d_{\mathrm{DGJP}}\,(x_1, x_2)$.

Proof.

$$\begin{aligned}
&d_{\mathrm{H}}\,(x_1, x_2)\\
&= d_{\mathit{fix}\,(F)}\,([\![x_1]\!]_{\langle X,\mu\rangle}, [\![x_2]\!]_{\langle X,\mu\rangle})\\
&= d_{\mathit{fix}\,(F)}\,([\![x_1]\!]_{\langle X,\mu'\rangle}, [\![x_2]\!]_{\langle X,\mu'\rangle}) \quad [(2)]\\
&\leq d_{\mathrm{DGJP}}\,(x_1, x_2) \quad [[\![-]\!]_{\langle X,\mu'\rangle} \text{ is nonexpansive}]
\end{aligned}$$

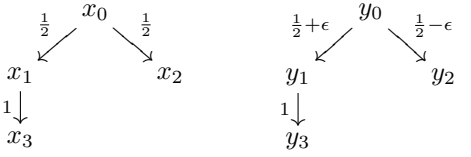
□

Thus we have shown that our pseudometric can also be characterized by a real-valued modal logic similar to the one studied by Desharnais et al.

4 De Vink and Rutten

We make another comparison, this time with the distance function introduced by De Vink and Rutten in [21]. Remarks similar to the ones made below about their distance function apply also to the distance functions presented by Baier and Kwiatkowska [3] and Den Hartog [12].

Consider the following two probabilistic transition systems.



Clearly, the smaller ϵ is, the more alike these systems behave. Our pseudometric captures this since $d_{\mathrm{H}}\,(x_0, y_0) = \frac{\epsilon}{4}$. However, in De Vink and Rutten's setting these systems are $\frac{1}{2}$ apart if $\epsilon \neq 0$. More generally, the distance between two systems in their setting is 2^{-n-1} where n is the depth of probabilistic bisimilarity between them.

De Vink and Rutten consider the functor

$$G = \mathbf{1} + \mathcal{M}_c\,(\tfrac{1}{2} \cdot -) : \mathcal{CUM}et_1 \to \mathcal{CUM}et_1,$$

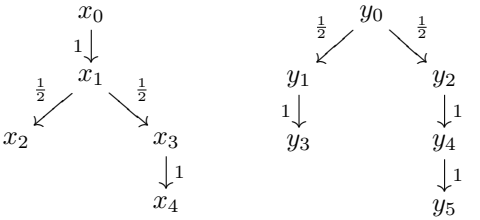
where $\mathcal{M}_c$ denotes the Borel probability measures with compact support. The main differences between our functor F and their functor G are the following.

- They consider a distance function on Borel probability measures [21, Definition 5.3] different from the one of Hutchinson (Definition 1). Their distance function only captures qualitative information as the above example illustrates.
- They consider the category $\mathcal{CUM}et_1$ of 1-bounded complete ultrametric spaces and nonexpansive functions whereas we consider the considerably larger category $\mathcal{PM}et_1$. This allows us to captures many more interesting continuous probabilistic transition systems as coalgebras, including systems where the state space is the real interval $[0,1]$ endowed with the Euclidean metric.
- They consider Borel probability measures with compact support whereas we consider the more general tight Borel probability measures. Again this allows us to represent more systems as coalgebras.
- Their model only allows processes to refuse transitions with probability 0 or 1.

We have generalized all the results for the functor G in [21, Section 5][9] to our setting.

5 Norman

We compare our pseudometric with the pseudometric introduced by Norman in [17, Section 6.1]. Consider the following two probabilistic transition systems.



These systems are not probabilistic bisimilar. In Norman's pseudometric the systems have distance 0. In our pseudometric, systems only have distance 0 if they are probabilistic bisimilar. In our setting the systems are $\frac{1}{16}$ apart. This example also shows that his pseudometric gives rise to a topology different from ours.

The main differences between his and our pseudometric are the following.

- He uses a linear-time model whereas we consider a branching-time model.
- He considers only discrete probabilistic transition systems whereas we also consider continuous ones.
- We use the usual categorical machinery and various standard constructions whereas his definitions are more ad-hoc. We believe however that his pseudometric can also be characterized by means of a terminal coalgebra.

[9] The proof of [21, Theorem 5.8] is incomplete. We also have no proof for this result in our setting.

Conclusion

In this paper, we have presented a new pseudometric on a class of probabilistic transitions systems. The pseudometric was defined via the terminal coalgebra of a functor based on the Hutchinson metric on the space of Borel probability measures on a pseudometric space. We also characterized the distance between systems in terms of a real-valued modal logic. Similar results have been presented by the second author in his thesis [22] in the setting of bimodules and generalized metric spaces.

Let us isolate two distinct consequences of our use of the Hutchinson metric. We can talk about approximate equivalence of processes and we can model continuous-state systems as coalgebras. An apparent restriction with regard to the latter point is the requirement that the structure map of an F-coalgebra be nonexpansive. Properly speaking, continuous probabilistic transition systems as formulated in Section 2 are coalgebras of (a variant of) the Giry monad on the category of measurable spaces [11]. However, we conjecture that the terminal F-coalgebra $\langle \mathit{fix}\,(F), \iota \rangle$is also terminal when seen as a coalgebra of the Giry functor, and that our results can be extended to continuous-state systems in general.

Exploiting Theorem 1 and some results by Alessi et al. [2] we have shown that our terminal coalgebra is compact and hence separable. Furthermore we have shown that the unique map from the initial algebra of a finitary version of F—representing finite discrete probabilistic transition systems with rational probabilities—to the terminal F-coalgebra is a dense embedding. Hence, every continuous system can be approximated by a finite one (see also [8]).

Making use of linear programming, we have developed an algorithm that calculates our distance between finite state systems to a prescribed degree of accuracy in polynomial time, cf. [4].

Many system combinators can be shown to be nonexpansive with respect to our pseudometric. This quantitative analogue of congruence allows for compositional verification (see also [7,10]).

Acknowledgements. The authors would like to thank the Amsterdam Coordination Group, Josée Desharnais, Abbas Edalat, Joel Ouaknine, Prakash Panangaden, Jan Rutten and Erik de Vink for discussion. The first author is thankful to Stephen Watson for the joint study of the Hutchinson metric.

References

1. R.B. Ash. *Real Analysis and Probability*, Academic Press, London, 1972.
2. F. Alessi, P. Baldan and G. Bellè. A Fixed-Point Theorem in a Category of Compact Metric Spaces. *Theoretical Computer Science*, 146(1/2):311-320, July 1995.
3. C. Baier and M. Kwiatkowska. Domain Equations for Probabilistic Processes. In *Proceedings of the 4th International Workshop on Expressiveness in Concurrency*, volume 7 of Electronic Notes in Theoretical Computer Science, Santa Margherita Ligure, September 1997, Elsevier.

4. F. van Breugel and J. Worrell. An Algorithm for Quantitative Verification of Probabilistic Transition Systems. Report CS-2001-01, York University, Toronto, April 2001.
5. J. Desharnais, A. Edalat and P. Panangaden. A Logical Characterization of Bisimulation for Labelled Markov Processes. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, pages 478-487, Indianapolis, 1988. IEEE.
6. J. Desharnais, A. Edalat and P. Panangaden. Bisimulation for Labelled Markov Processes. *Information and Computation*, to appear.
7. J. Desharnais, V. Gupta, R. Jagadeesan and P. Panangaden. Metrics for Labelled Markov Systems. In *Proceedings of the 10th International Conference on Concurrency Theory*, vol. 1664 of Lect. Notes in Comp. Sci, pages 258-273, Eindhoven, August 1999. Springer-Verlag.
8. J. Desharnais, V. Gupta, R. Jagadeesan and P. Panangaden. Approximating Labelled Markov Processes. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*, pages 95-106, Santa Barbara, June 2000. IEEE.
9. G.A. Edgar. *Integral, Probability, and Fractal Measures*, Springer-Verlag, 1998.
10. A. Giacalone, C.C. Jou and S.A. Smolka. Algebraic Reasoning for Probabilistic Concurrent Systems. In *Proceedings of the IFIP WG 2.2/2.3 Working Conference on Programming Concepts and Methods*, pages 443-458, Sea of Galilee, April 1990, North-Holland.
11. M. Giry. A Categorical Approach to Probability Theory. In *Proceedings of the International Conference on Categorical Aspects of Topology and Analysis*, volume 915 of Lect. Notes in Math., pages 68-85, Ottawa, 1981, Springer-Verlag.
12. J.I. den Hartog. *Comparative Semantics for a Process Language with Probabilistic Choice and Non-Determinism*, Report IR-445, Free University, Amsterdam, 1998.
13. M. Huth and M. Kwiatkowska. Quantitative Analysis and Model Checking. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, pages 111-122, Warsaw, June/July 1997. IEEE.
14. J.E. Hutchinson. Fractals and Self Similarity. *Indiana University Mathematics Journal*, 30(5):713-747, 1981.
15. D. Kozen. A Probabilistic PDL. *Journal of Computer and System Sciences*, 30(2):162-178, April 1985.
16. K.G. Larsen and A. Skou. Bisimulation through Probabilistic Testing. *Information and Computation*, 94(1):1-28, September 1991.
17. G.J. Norman. *Metric Semantics for Reactive Probabilistic Systems*. PhD thesis, University of Birmingham, 1997.
18. K.R. Parthasarathy. *Probability Measures on Metric Spaces*. Academic Press, 1967.
19. J.J.J.M. Rutten. Universal Coalgebra: a Theory of Systems, *Theoretical Computer Science*, 249(1), pages 3-80, October 2000.
20. D. Turi and J.J.M.M. Rutten. On the Foundations of Final Semantics: non-standard sets, metric spaces, partial orders. *Mathematical Structures in Computer Science*, 8(5):481-540, October 1998.
21. E.P. de Vink and J.J.M.M. Rutten. Bisimulation for Probabilistic Transition Systems: a Coalgebraic Approach. *Theoretical Computer Science*, 221(1/2):271-293, June 1999.
22. J. Worrell. *On Coalgebras and Final Semantics*. PhD thesis, Oxford University, 2000.

Efficient Generation of Plane Triangulations without Repetitions

Zhangjian Li and Shin-ichi Nakano

Gunma University, Kiryu 376-8515, Japan,
kenlee@msc.cs.gunma-u.ac.jp, nakano@cs.gunma-u.ac.jp
Fax: +81-277-30-1812

Abstract. A "based" plane triangulation is a plane triangulation with one designated edge on the outer face. In this paper we give a simple algorithm to generate all biconnected based plane triangulations with at most n vertices. The algorithm uses $O(n)$ space and generates such triangulations in $O(1)$ time per triangulation without duplications. The algorithm does not output entire triangulations but the difference from the previous triangulation. By modifying the algorithm we can generate all biconnected based plane triangulation having exactly n vertices including exactly r vertices on the outer face in $O(1)$ time per triangulation without duplications, while the previous best algorithm generates such triangulations in $O(n^2)$ time per triangulation. Also we can generate without duplications all biconnected (non-based) plane triangulations having exactly n vertices including exactly r vertices on the outer face in $O(r^2n)$ time per triangulation, and all maximal planar graphs having exactly n vertices in $O(n^3)$ time per graph.

1 Introduction

Generating all graphs with some property without duplications has many applications, including unbiased statistical analysis[M98]. A lot of algorithms to solve these problems are already known [A96,B80,M98,W86, etc]. See nice textbooks [G93,KS98].

In this paper we wish to generate all biconnected "based" plane triangulations, which will be defined precisely in Section 2, with at most n vertices. Such triangulations play an important role in many algorithms, including graph drawing algorithms [CN98,FPP90,S90, etc].

To solve these all-graph-generating problems some types of algorithms are known.

Classical method algorithms [G93, p57] first generate all the graphs with given property allowing duplications, but output only if the graph has not been output yet. Thus this method requires quite a huge space to store a list of graphs that have already been output. Furthermore, checking whether each graph has already been output requires a lot of time.

Orderly method algorithms [G93, p57] need not to store the list, since they output a graph only if it is a "canonical" representative of each isomorphism class.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 433–443, 2001.

Reverse search method algorithms [A96] also need not to store the list. The idea is to implicitly define a connected graph H such that the vertices of H correspond to the graphs with the given property, and the edges of H correspond to some relation between the graphs. By traversing an implicitly defined spanning tree of H, one can find all the vertices of H, which correspond to all the graphs with the given property.

The main idea of our algorithm is that for some problems we can define a tree (not a general graph) as the graph H of reverse search method. Thus our algorithm does not need to find a spanning tree of H, since H itself is a tree. With some other ideas we give the following four simple but efficient algorithms.

Our first algorithm generates all biconnected based plane triangulations with at most n vertices. A *based* plane triangulation means a plane triangulation with one designated "base" edge on the outer face. For instance there are four biconnected based plane triangulations with at most four vertices, as shown in Fig. 1(a). The base edges are depicted by thick lines. However, there are only three biconnected plane triangulations with at most four vertices. See Fig. 1(b). The algorithm uses $O(n)$ space and runs in $O(f(n))$ time, where $f(n)$ is the number of nonisomorphic biconnected based plane triangulations with at most n vertices. The algorithm generates triangulations without duplications. So the algorithm generates each triangulation in $O(1)$ time on average. The algorithm does not output entire triangulations but the difference from the previous triangulation.

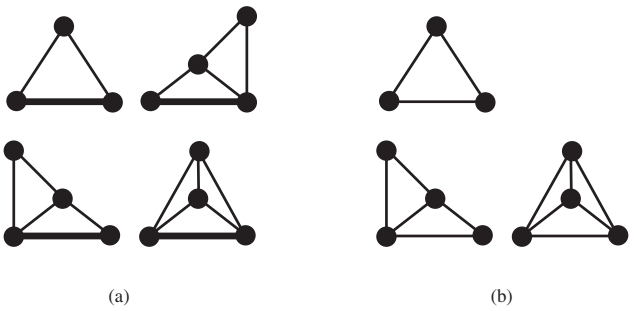


Fig. 1. (a) Biconnected based plane triangulations, and (b) biconnected plane triangulations.

By modifying our first algorithm we can generate without duplications all biconnected based plane triangulations having exactly n vertices including exactly r vertices on the outer face. The algorithm uses $O(n)$ space and runs in $O(f(n,r))$ time, where $f(n,r)$ is the number of nonisomorphic such triangulations. So the algorithm generates each triangulation in $O(1)$ time on average, while the previous best algorithm [A96] generates such triangulations in $O(n^2)$ time per triangulation.

Also we can generate all biconnected (non-based) plane triangulations having exactly n vertices including exactly r vertices on the outer face in $O(r^2n)$ time (on average) per triangulation. Another algorithm with $O(n^2)$ time per triangulation is also claimed in [M98] without detail but using a complicated theoretical linear-time plane graph isomorphism algorithm [HW74], while our algorithm is simple and does not need the isomorphism algorithm.

Also we can generate all maximal planar graphs having exactly n vertices in $O(n^3)$ time (on average) per graph.

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 shows a tree structure among biconnected based plane triangulations. Section 4 presents our first algorithm. By modifying the algorithm we give three more algorithms in Section 5. Finally Section 6 is a conclusion.

2 Preliminaries

In this section we give some definitions.

Let G be a connected graph with n vertices. An edge connecting vertices x and y is denoted by (x, y). The *degree* of a vertex v is the number of neighbors of v in G. A *cut* is a set of vertices whose removal results in a disconnected graph or a single-vertex graph K_1. The *connectivity* $\kappa(G)$ of a graph G is the cardinality of the minimum number of vertices consisting a cut. G is $k-$*connected* if $k \leq \kappa(G)$.

A graph is *planar* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. A *plane* graph is a planar graph with a fixed planar embedding. A plane graph divides the plane into connected regions called *faces*. The unbounded face is called *the outer face*, and other faces are called *inner faces*. We regard *the contour* of a face as the clockwise cycle formed by the vertices and edges on the boundary of the face. We denote the contour of the outer face of plane graph G by $C_o(G)$. A plane graph is called *a plane triangulation* if each inner face has exactly three edges on its contour. A *based* plane triangulation is a plane triangulation with one designated edge on the contour of the outer face. The designated edge is called *the base edge*.

3 The Removing Sequence and the Genealogical Tree

Let S_n be the set of all biconnected based plane triangulations with at most n vertices. In this section we explain a tree structure among the triangulations in S_n.

Let G be a biconnected based plane triangulation having four or more vertices. Let $C_o(G) = w_1, w_2, \cdots, w_k$, and (w_1, w_k) be the base edge of G.

A vertex $w_s, 1 < s < k$, on $C_o(G)$ is *removable* if removing w_s from G preserves biconnectivity. Since G is a biconnected based plane triangulation, the resulting graph after removing a removable vertex v is also a biconnected based plane triangulation with the same base edge.

An edge (w_i, w_j) in G is called a *chord* of G if $i + 2 \leq j$. Intuitively, each chord is an edge connecting two non-consecutive vertices on $C_o(G)$. However, especially, the base edge (w_1, w_k) is also a chord. So G always has at least one chord.

We have the following lemma.

Lemma 1. *Every biconnected based plane triangulation with four or more vertices has at least one removable vertex.*

Proof. Let G be a biconnected based plane triangulation having four or more vertices. Let (w_i, w_j) be a chord with the minimum $j - i$, where $i < j$. Then each $w_s, i < s < j$, is removable, because no cut consisting of exactly one vertex appears after removing w_s. □

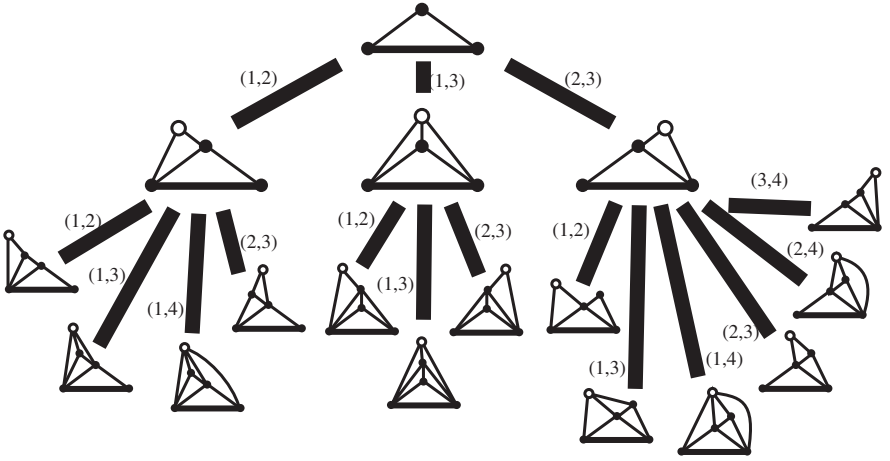


Fig. 2. Genealogical tree T_5.

If w_s is removable but $w_2, w_3, \cdots, w_{s-1}$ are not, then w_s is called *the leftmost removable vertex* of G. We can observe that if w_s is the leftmost removable vertex then each of $w_2, w_3, \cdots, w_{s-1}$ is an end of at least one chord. (So they are not removable.)

For each triangulation G in S_n except K_3, if we remove the leftmost removable vertex then the resulting triangulation, denoted by $P(G)$, is also a triangulation in S_n having one less vertices. Thus we can define the unique triangulation $P(G)$ in S_n for each G in S_n except K_3. We say G is a *child* triangulation of $P(G)$.

Given a triangulation G in S_n, by repeatedly removing the leftmost removable vertex, we can have the unique sequence $G, P(G), P(P(G)), \cdots$ of triangulations in S_n which eventually ends with K_3. By merging those sequences we can have *the genealogical tree* T_n of S_n such that the vertices of T_n correspond to the

triangulations in S_n, and each edge corresponds to each relation between some G and $P(G)$. For instance T_5 is shown in Fig. 2, in which each leftmost removable vertex is depicted by a white circle. We call the vertex in T_n corresponding to K_3 *the root* of T_n.

4 Algorithm

Given S_n we can construct T_n by the definition, possibly with a huge space and much running time. However, how can we construct T_n efficiently only given an integer n? Our idea is by reversing the removing procedure as follows.

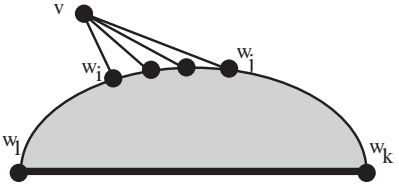


Fig. 3. Illustration for $G(i, j)$.

Given a biconnected based plane triangulation G in S_n with at most $n-1$ vertices, we wish to find all child triangulations of G. Let $C_o(G) = w_1, w_2, \cdots, w_k$, and (w_1, w_k) be the base edge of G, and w_s be the leftmost removable vertex of G. Since K_3 has no removable vertex, for convenience, we regard $w_k(= w_3)$ as the leftmost removable vertex for K_3. We denote by $G(i, j)$, $1 \leq i < j \leq k$, the based plane triangulation obtained from G by adding new vertex v on the outer face of G, and adding $j - i + 1 \geq 2$ edges $(w_i, v), (w_{i+1}, v), \cdots, (w_j, v)$, as shown in Fig. 3. $G(i, j)$ is a child triangulation of G if and only if v is the leftmost removable vertex of $G(i, j)$.

Since w_s is the leftmost removable vertex of G, each w_t, $1 < t < s$, has at least one chord (w_t, w_u) such that $s < u$. (Otherwise, w_s is not the leftmost removable, a contradiction.) We denote by $q(t)$ the largest index such that $(w_t, w_{q(t)})$ is a chord.

We have the following four cases to consider.

Case 1: $j \leq s$.

In this case v is the leftmost removable vertex of $G(i, j)$. Thus $P(G(i, j)) = G$.

Case 2: $i < s < j$.

If $j > w_{q(i)}$, then w_i not v is the leftmost removable vertex of $G(i, j)$, and $P(G(i, j)) \neq G$. Otherwise, v is the leftmost removable vertex of $G(i, j)$, and $P(G(i, j)) = G$.

Case 3: $i = s$.

If $j = i+1$ then v is the leftmost removable vertex of $G(i, j)$, and $P(G(i, j)) = G$. Otherwise, $j \geq i + 2$ holds then w_s is a (possibly leftmost) removable vertex of $G(i, j)$, and $P(G(i, j)) \neq G$.

Case 4: $i > s$.

In this case v is not the leftmost removable vertex of $G(i,j)$. Thus $P(G(i,j)) \neq G$.

Based on the case analysis above we can find all child triangulations of given triangulation in S_n. If G has k child triangulations, then we can find them in $O(k)$ time. This is an intuitive reason why our algorithm generates triangulations in $O(1)$ time per triangulation.

And recursively repeating this process from the root of T_n corresponding to K_3 we can traverse T_n without constructing whole T_n. During the traversing of T_n, we assign a label (i,j) to each edge connecting G and $G(i,j)$ in T_n, as shown in Fig. 2. Each label denotes how to add a new vertex to G to generate a child triangulation $G(i,j)$, and each sequence of labels on a path starting from the root specifies a triangulation in S_n. For instance $(1,2),(1,2)$ specify the leftmost triangulation in Fig. 2. During our algorithm we will maintain these labels only on the path from the root to the "current" vertex, because those are enough information to generate the "current" triangulation. To generate next triangulation, we need to maintain some more information (the leftmost removable vertex w_s, and $w_{q(t)}$ for each $1 < t < s$, etc.) only for the triangulations on the "current" path, which has length at most n. This is an intuitive reason why our algorithm uses only $O(n)$ space, while the number of triangulations may not be bounded by a polynomial in n.

Our algorithm is as follows.

```
Procedure find-all-child-triangulations(G)
begin
1   output G { Output the difference from the previous triangulation}
2   if G has exactly n vertices then return
3   for i = 1 to s − 1
4     for j = i + 1 to s
5       find-all-child-triangulations(G(i, j))          { Case 1}
6   for i = 1 to s − 1
7     for j = s + 1 to q(i)
8       find-all-child-triangulations(G(i, j))          { Case 2}
9   find-all-child-triangulations(G(s, s + 1))          { Case 3}
end

Algorithm find-all-triangulations(T3)
begin
1   output K3
2   G = K3
3   find-all-child-triangulations(G(1, 2))
4   find-all-child-triangulations(G(2, 3))
5   find-all-child-triangulations(G(1, 3))
end
```

Theorem 1. *The algorithm uses $O(n)$ space and runs in $O(f(n))$ time, where $f(n)$ is the number of nonisomorphic biconnected based plane triangulations with at most n vertices.*

Proof. We need to maintain for current triangulation (i) a doubly linked list of vertices on C_o, (ii) the leftmost removable vertex w_s, and (iii) $w_{q(t)}$ for each $1 < t < s$. When we recursively call the **find-all-child-triangulation**, we need to update the (i)–(iii) above, and when we return from the recursive call we need to restore the (i)–(iii) above. We can do these in (1) time, respectively, as follows.

We can update (i) easily.

When we recursively call, one of **Case 1**–**3** occurs, and then the newly added vertex always becomes the leftmost removable vertex of $G(i, j)$. Also by recoding this update on a stack we can restore (ii) when return occurs. Thus we can update (ii), too.

Also, when we recursively call, if either **Case 1** or **2** occurs, then we already have all (iii), since (iii) of $G(i, j)$ is a prefix of (iii) of G, otherwise **Case 3** occurs, then we only need to set the w_{s+1} of G as $w_{q(s)}$ of $G(i, j)$. Again by recoding this update on a stack we can restore (iii) when return occurs.

Thus we can update (i)–(iii) in $O(1)$ time.

For other part our algorithm needs only a constant time of computations for each edge of the tree. Thus the algorithm runs in $O(f(n))$ time.

For each recursive call we need a constant number of space, and the depth of recursive call is bounded by n. Thus the algorithm uses $O(n)$ space. □

5 Modification of the Algorithm

Then we consider our second problem.

A vertex v of G is called *an inner vertex* of G if v is not on $C_o(G)$. Let S_{n-1}^{n-r} be the set of biconnected based plane triangulation having at most $n-1$ vertices including at most $n-r$ inner vertices. And let $S_{=n}^{=n-r}$ be the set of biconnected based plane triangulation having exactly n vertices including exactly $n-r$ inner vertices.

We wish to generate all triangulations in $S_{=n}^{=n-r}$ without duplications.

For each triangulation G in $S_{=n}^{=n-r}$ if we remove the leftmost removable vertex v then the resulting triangulation $P(G)$ is a triangulation in S_{n-1}^{n-r} having one less vertices, and if v has exactly two neighbors on $C_o(P(G))$, then G has the same number of inner vertices with $P(G)$, otherwise v has three or more neighbors on $C_o(P(G))$, and then G has more inner vertices than $P(G)$ has.

Also, for each triangulation G in S_{n-1}^{n-r} except K_3 if we remove the leftmost removable vertex then the resulting triangulation $P(G)$ is also a triangulation in S_{n-1}^{n-r} having one less vertices and having less or equal number of inner vertices.

Thus for each G in $S_{=n}^{=n-r} \cup S_{n-1}^{n-r}$ except K_3 we can again define the unique triangulation $P(G)$ in S_{n-1}^{n-r}. Thus we again have *the genealogical tree* T_n^{n-r} such that (i) the leaf vertices of T_n^{n-r} correspond to the triangulations in $S_{=n}^{=n-r}$, (ii)

the non-leaf vertices of T_n^{n-r} correspond to the triangulations in S_{n-1}^{n-r}, and (iii) each edge corresponds to each relation between some G and $P(G)$. For instance T_5^{5-4} is shown in Fig. 4, in which each leftmost removable vertex is depicted by a white circle. The vertex in T_n^{n-r} corresponding to K_3 is called *the root* of T_n^{n-r}.

Given a triangulation G in S_{n-1}^{n-r}, we wish to find all child triangulations of G. Let $C_o(G) = w_1, w_2, \cdots, w_k$, and (w_1, w_k) be the base edge of G, and w_s be the leftmost removable vertex of G. We denote by $G(i, j)$, $1 \leq i < j \leq k$, the based plane triangulation obtained from G by adding new vertex v on the outer face of G, and adding $j - i + 1 \geq 2$ edges $(w_i, v), (w_{i+1}, v), \cdots, (w_j, v)$, as shown in Fig. 3.

We have the following lemma.

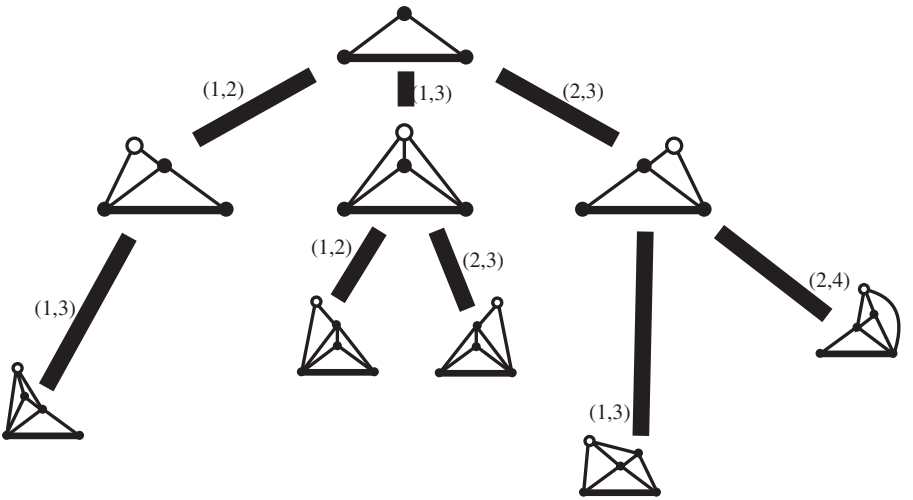


Fig. 4. Genealogical tree T_5^{5-4}.

Lemma 2. *Let G be a based plane triangulation in S_{n-1}^{n-r}. (a) If G has at most $n-2$ vertices then G has at least two child triangulations in S_{n-1}^{n-r}. (b) If G has exactly $n-1$ vertices then G has at least one child triangulation in $S_{=n}^{n-r}$.*

Proof. If $G = K_3$ then the claim holds. Assume otherwise.

Let $C_o(G) = w_1, w_2, \cdots, w_k$, and (w_1, w_k) be the base edge of G. Let w_s be the leftmost removable vertex of G.

(a) $G(s-1, s)$ and $G(s, s+1)$ are child triangulations of G and in S_{n-1}^{n-r}.

(b) Let t be the number of inner vertex of G. By the definition of T_n^{n-r}, $t \leq n - r$ holds. Any child triangulation of G must have exactly $n - r$ inner vertices by the definition of T_n^{n-r}. Thus we have to add a new vertex to G with exactly $n - r - t + 2$ edges to have $n - r - t$ more inner vertices. Since v is the leftmost removable vertex of $G(1, n - r - t + 2)$, $G(1, n - r - t + 2)$ is a child triangulations of G and in $S_{=n}^{n-r}$. □

Now we wish to find all child triangulations of G. We have the following two cases to consider.

Let $C_o(G) = w_1, w_2, \cdots, w_k$, and (w_1, w_k) be the base edge of G, and w_s be the leftmost removable vertex of G. Let t be the number of inner vertices of G.
Case 1: G has exactly $n-1$ vertices.

If $t = n - r$ then only $G(i, i+1)$ such that $1 \leq i \leq s$ is a child triangulation of G.

Otherwise $t < n-r$ holds, and we need to add a new vertex to G with exactly $n-r-t+2$ edges to have $n-r-t > 0$ more inner vertices. Now only $G(i, j)$ such that (i) $i \leq s$, (ii) $j \leq q(i)$, and (iii) $j - i - 1 = n - r - t$, is a child triangulation of G. (If $i > s$ or $j > q(i)$ then the new vertex v is not the leftmost removable vertex of G. And if $j - i - 1 \neq n - r - t$ then the resulting graph cannot have exactly $n - r$ inner vertices.)
Case 2: G has at most $n-2$ vertices.

If $t = n - r$ then only $G(i, i+1)$ such that $1 \leq i \leq s$ is a child triangulation of G.

Otherwise $t < n - r$ holds, and we need to preserve the number of inner vertices at most $n - r$ after adding a new vertex to G.

We have the following four subcases similar to the four cases in Section 4.
Case 2(a): $j \leq s$.

If $t + (j - i - 1) \leq n - r$ then $P(G(i, j)) = G$, otherwise $P(G(i, j)) \neq G$.
Case 2(b): $i < s < j$.

If $j \leq q(i)$ and $t + (j - i - 1) \leq n - r$ then $P(G(i, j)) = G$, otherwise $P(G(i, j)) \neq G$.
Case 2(c): $i = s$.

If $j = i+1$ (now $t+(j-i-1) \leq n-r$ holds) then $P(G(i, j)) = G$, otherwise $P(G(i, j)) \neq G$.
Case 2(d): $i > s$. In this case $P(G(i, j)) \neq G$.

Based on the case analysis above we can have an algorithm to find all triangulations in $S_{=n}^{=n-r}$. We have the following lemma.

Lemma 3. *The algorithm uses $O(n)$ space and runs in $O(f(n, r))$ time, where $f(n, r)$ is the number of nonisomorphic biconnected based plane triangulations having exactly n vertices including exactly r vertices on the outer face.*

Proof. By Lemma 2 the number of vertices in T_n^{n-r} is at most $3 \cdot |S_{=n}^{=n-r}| = 3 \cdot f(n, r)$. And the algorithm need only a constant time of computation for each edge of T_n^{n-r}. Thus the algorithm runs in $O(f(n, r))$ time. The algorithm clearly uses $O(n)$ space. □

We modify our second algorithm so that it output all biconnected (non-based) plane triangulations having exactly n vertices including exactly r vertices on the outer face, as follows.

At each leaf v of the genealogical tree T_n^{n-r}, the triangulation G corresponding to v is checked whether the adding sequence of G with the base edge is the lexicographically first one among the r adding sequences of G for r choice of

the base edge on $C_o(G)$, and only if so G is output. Thus we can output only canonical representative of each isomorphism class.

Lemma 4. *The algorithm uses $O(n)$ space and runs in $O(r^2 n \cdot g(n,r))$ time, where $g(n,r)$ is the number of nonisomorphic biconnected (non-based) plane triangulations having exactly n vertices including exactly r vertices on the outer face.*

Proof. Given a biconnected based plane triangulation G, by counting (and updating) the number of chord incident to each vertex on $C_o(G)$, we can find the adding sequence in $O(n)$ time. For each triangulation corresponding to a leaf of T_n^{n-r}, we construct r adding sequences for r choice of the base edge on $C_o(G)$, and find the lexicographically first one in $O(rn)$ time, and for each output triangulation our tree may contain r isomorphic ones corresponding to r choices of the base edge. Thus the algorithm runs in $O(r^2 n \cdot g(n,r))$ time. The algorithm clearly uses $O(n)$ space. □

A planar graph with n vertices is *maximal* if it has exactly $3n-6$ edges. Every maximal planar graph except K_3 is triconnected, and every triconnected planar graph has a unique embedding on a sphere only up to mirror copy [HW74]. And in the embeddig every face has exactly three edges on its contour. Thus, for every maximal planar graph by choosing the outer face and the base edge, there are exactly $2m$ (biconnected) based plane triangulation G with exactly 3 vertices on the outer face, where m is the number of edges of G. We modify the algorithm further as follows. At each leaf v of the genealogical tree T_n^{n-3}, the triangulation G corresponding to v is checked whether the adding sequence of G with the base edge is the lexicographically first one among the $2m$ adding sequences of G (3 choice of the base edge on $C_o(G)$ for each of $2m/3$ choice of the outer face of G), and only if so G is output. Thus we have the following theorem, which is an answer for an open problem in [A96].

Theorem 2. *The modified algorithm generate all maximal planar graphs in $O(n^3 \cdot h(n))$ time, where $h(n)$ is the number of nonisomorphic maximal planar graphs with exactly n vertices. The algorithm uses $O(n)$ space.*

6 Conclusion

In this paper we have given four simple algorithms to generate all graphs with some property. Our algorithms first define a genealogical tree such that each vertex corresponds to each graph of the given property, then output each graph without duplications by traversing the tree.

To find other all-something-generating problems to which our method can be applied is remained as an open problem.

References

[A96] D. Avis, *Generating rooted triangulations without repetitions*, Algorithmica, 16, (1996), pp.618-632.

[B80] T. Beyer and S. M. Hedetniemi, *Constant time generation of rooted trees*, SIAM J. Comput., 9, (1980), pp.706-712.

[CN98] M. Chrobak and S. Nakano, *Minimum-width grid drawings of plane graphs*, Computational Geometry: Theory and Applications, 10, (1998), pp.29-54.

[FPP90] H. de Fraysseix, J. Pach and R. Pollack, *How to draw a planar graph on a grid*, Combinatorica, 10, (1990), pp.41-51.

[G93] L. A. Goldberg, *Efficient algorithms for listing combinatorial structures*, Cambridge University Press, New York, (1993).

[HW74] J. E. Hopcroft and J.K. Wong, *Linear time algorithm for isomorphism of planar graphs*, Proc. of 6th STOC, (1974), pp.172-184.

[KS98] D. L. Kreher and D. R. Stinson, *Combinatorial algorithms*, CRC Press, Boca Raton, (1998).

[M98] B. D. McKay, *Isomorph-free exhaustive generation*, J. of Algorithms, 26, (1998), pp.306-324.

[S90] W. Schnyder, *Embedding planar graphs on the grid*, Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms, San Francisco, (1990), pp.138-148.

[W86] R. A. Wright, B. Richmond, A. Odlyzko and B. D. McKay, *Constant time generation of free trees*, SIAM J. Comput., 15, (1986), pp.540-548.

The Longest Common Subsequence Problem for Sequences with Nested Arc Annotations
(Extended Abstract)

Guo-Hui Lin[1,2] *, Zhi-Zhong Chen[3] **, Tao Jiang[2,4] ***, and Jianjun Wen[4] †

[1] Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada.
[2] Department of Computing and Software, McMaster University, Hamilton, Ontario L8S 4L7, Canada.
[3] Department of Mathematical Sciences, Tokyo Denki University, Hatoyama, Saitama 350-0394, Japan.
[4] Department of Computer Science, University of California, Riverside, CA 92521.

Abstract. Arc-annotated sequences are useful in representing the structural information of RNA and protein sequences. The Longest Arc-Preserving Common Subsequence (LAPCS) Problem has been introduced in [11] as a framework for studying the similarity of arc-annotated sequences. Several algorithmic and complexity results on the LAPCS problem have been presented in [11,17]. In this paper, we continue this line of research and present new algorithmic and complexity results on the LAPCS problem restricted to two nested arc-annotated sequences, denoted as LAPCS(nested, nested). The restricted problem is perhaps the most interesting variant of the LAPCS problem and has important applications in the comparison of RNA secondary and tertiary structures. Particularly, we prove that LAPCS(nested, nested) is NP-hard, which answers an open question in [11]. We then present a polynomial-time approximation scheme for LAPCS(nested, nested) with an additional *c-diagonal* restriction. An interesting special case, unary LAPCS(nested, nested), is also investigated.

1 Introduction

Given two sequences S and T over some fixed *alphabet* Σ, sequence T is said to be a *subsequence* of S if T can be obtained from S by deleting some letters

* Supported in part by NSERC Research Grant OGP0046613 and a CITO grant. ghlin@math.uwaterloo.ca.

** Supported in part by the Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan, under Grant No. 12780241. chen@r.dendai.ac.jp. Work done while visiting at UC Riverside.

*** Supported in part by a UCR startup grant and NSF Grants CCR-9988353 and ITR-0085910. jiang@cs.ucr.edu.

† Supported in part by NSF Grant CCR-9988353. wjianju@cs.ucr.edu.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 444–455, 2001.

(also called *bases*) from S. Notice that the order of the remaining letters of S must be preserved. The *length* of a sequence S, denoted by $|S|$, is the number of letters in S. Given two sequences S_1 and S_2 (over some fixed *alphabet* Σ), the classical Longest Common Subsequence (LCS) problem asks for a longest sequence T that is a subsequence of both S_1 and S_2. Suppose $|S_1| = n_1$ and $|S_2| = n_2$, then a longest common subsequence of S_1 and S_2 can be computed by dynamic programming in time $O(n_1 n_2)$ [16,23,24]. For simplicity, we use $S[i]$ to denote the ith letter in sequence S, and $S[i_1, i_2]$ $(1 \leq i_1 \leq i_2 \leq |S|)$ to denote the substring of S consisting of the i_1th letter through the i_2th letter.

For any sequence S, an *arc annotation* set (or simply an arc set) P of S is a set of *unordered* pairs of positions in S. Each pair $(i_1, i_2) \in P$, where $1 \leq i_1 < i_2 \leq |S|$, is said to *connect* the two letters at positions i_1 and i_2 and is called an *arc annotation* (or simply, arc) between the two letters. Such a pair (S, P) of sequence and arc annotation set is referred to as an *arc-annotated* sequence [11]. Observe that a (plain) sequence without any arc can be viewed as an arc-annotated sequence with an empty arc set.

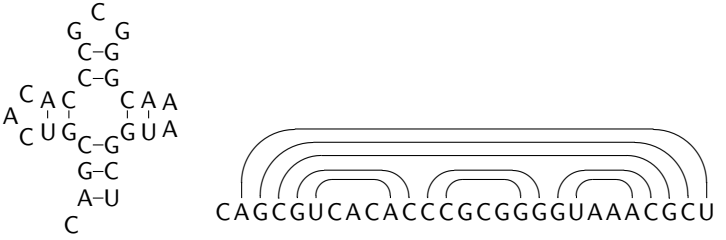


Fig. 1. A tRNA and its corresponding arc-annotated sequence.

Arc-annotated sequences are useful in describing the secondary and tertiary structures of RNA and protein sequences [2,10,11,14,18,22,26]. For example, one may use arcs to represent bonds between nucleotides in an RNA (more precisely, *transfer RNA* or tRNA) sequence (see Figure 1) or contact forces between amino acids in a protein sequence. Therefore, the problem of comparing arc-annotated sequences has applications in the structural comparison of RNA and protein sequences and has received much attention in the literature recently [2,11,14,17, 18,26]. In this paper, we follow the LCS approach proposed in [11] and study the Longest Arc-Preserving Common Subsequence Problem for arc-annotated sequences.

Given two arc-annotated sequences S_1 and S_2 with arc sets P_1 and P_2, respectively, if $S_1[i] = S_2[j]$ for some pair of integers i and j, we name the pair $\langle i, j \rangle$ a *base match*; and if $S_1[i_1] = S_2[j_1]$, $S_1[i_2] = S_2[j_2]$, $(i_1, i_2) \in P_1$, and $(j_1, j_2) \in P_2$, for some integers $i_1 < i_2$ and $j_1 < j_2$, we name the pair $\langle (i_1, i_2), (j_1, j_2) \rangle$ an *arc match*. A common subsequence T of S_1 and S_2 induces a bijective mapping from a subset of $\{1, 2, \ldots, n_1\}$ to a subset of $\{1, 2, \ldots, n_2\}$,

where $n_1 = |S_1|$ and $n_2 = |S_2|$. Let M denote this mapping and suppose that $M = \{\langle i_\ell, j_\ell \rangle, \ell = 1, \ldots, |T|\}$, then we say that T *induces* the base matches $\langle i_\ell, j_\ell \rangle$, $\ell = 1, \ldots, |T|$. The common subsequence T is *arc-preserving* if the arcs induced by the mapping are preserved, *i.e.* for any $\langle i_{\ell_1}, j_{\ell_1} \rangle$, $\langle i_{\ell_2}, j_{\ell_2} \rangle \in M$:

$$(i_{\ell_1}, i_{\ell_2}) \in P_1 \Longleftrightarrow (j_{\ell_1}, j_{\ell_2}) \in P_2;$$

and we say in this case that T induces the arc match $\langle (i_{\ell_1}, i_{\ell_2}), (j_{\ell_1}, j_{\ell_2}) \rangle$ if $(i_{\ell_1}, i_{\ell_2}) \in P_1$ and $(j_{\ell_1}, j_{\ell_2}) \in P_2$ in addition. The Longest Arc-Preserving Common Subsequence (LAPCS) problem is to find a longest common subsequence of S_1 and S_2 that is arc-preserving (with respect to the given arc sets P_1 and P_2) [11].

It is shown in [11] that the LAPCS problem is NP-hard, if the arc annotation structures are unrestricted. Since in the practice of RNA and protein sequence comparisons arc sets are likely to satisfy some constraints (*e.g.* bond arcs do not cross in the case of tRNA sequences), it is of interest to consider various restrictions on arc structures. The following four natural restrictions on an arc set P have been discussed in the literature [11].

1. No two arcs share an endpoint:
   $\forall\ (i_1, i_2), (i_3, i_4) \in P, \quad i_1 \neq i_4$, $i_2 \neq i_3$, and $i_1 = i_3 \Longleftrightarrow i_2 = i_4$.
2. No two arcs cross each other:
   $\forall\ (i_1, i_2), (i_3, i_4) \in P, \quad i_1 \in [i_3, i_4] \Longleftrightarrow i_2 \in [i_3, i_4]$.
3. No two arcs nest:
   $\forall\ (i_1, i_2), (i_3, i_4) \in P, \quad i_1 \leq i_3 \Longleftrightarrow i_2 \leq i_3$.
4. There are no arcs at all:
   $P = \emptyset$.

These restrictions are used progressively and inclusively to produce five distinct *levels* of permitted arc structures on the sequences in the LAPCS problem:

- unlimited — no restrictions.
- crossing — restriction 1.
- nested — restrictions 1 and 2.
- chain — restrictions 1, 2 and 3.
- plain — restriction 4.

In the following, we use the notation LAPCS(level-1, level-2) [17] to represent the LAPCS problem where the arc structure of sequence S_1 is of level level-1 and the arc structure of sequence S_2 is of level level-2. Without loss of generality, we always assume that level-1 is at the same level of or higher than level-2. Problem LAPCS(unlimited, level-2) is NP-hard [11] and is not approximable within ratio n^ϵ, $\epsilon \in (0, \frac{1}{4})$ [17], where $n = \max\{n_1, n_2\}$. Problem LAPCS(crossing, level-2) is also NP-hard [11], and is MAX SNP-hard [17], and admits a 2-approximation [17]. If level-1 is at most nested and level-2 is lower than nested, then LAPCS(level-1, level-2) is solvable in polynomial time [11,16,17,23,24]. Prior to this work, the most interesting

yet unsolved case is LAPCS(NESTED, NESTED), except that it inherits the 2-approximation algorithm designed for LAPCS(CROSSING, CROSSING) [17].

Notice that the problem LAPCS(NESTED, NESTED) effectively models the similarity between two tRNA sequences, particularly the secondary structures, and is generally thought of as the most important variant of the LAPCS problem. For example, the arc structure in Figure 1 is nested. In this paper, we investigate the computational complexity of this problem and show its NP-hardness, answering an open question in [11]. The hardness result in fact also holds for a rather special case, denoted by 2-FRAGMENTED LAPCS(NESTED, NESTED), where all base matches induced by an LAPCS are required to have the form $\langle 2i + \frac{1}{2} \pm \frac{1}{2}, 2i + \frac{1}{2} \pm \frac{1}{2} \rangle$. In the positive aspect, we present a polynomial-time approximation scheme (PTAS) for a much more general problem called c-DIAGONAL LAPCS(NESTED, NESTED) than 2-FRAGMENTED LAPCS(NESTED, NESTED). Here, for any constant $c \geq 1$, c-DIAGONAL LAPCS(NESTED, NESTED) is the special case of LAPCS(NESTED, NESTED) where base $S_1[i]$ (respectively, $S_2[j]$) is allowed only to match bases in the range $S_2[i-c, i+c]$ (respectively, $S_1[j-c, j+c]$). The c-diagonal restriction has been studied extensively for *sequence alignment* problems in the literature [15,19]. The c-DIAGONAL LAPCS(NESTED, NESTED) problem is relevant in the comparison of conserved tRNA sequences where we already have a rough idea about the correspondence between bases in the two sequences. The PTAS is based on an interesting application of the bounded treewidth decomposition technique for planar graphs due to Baker [3].

The rest of the paper is organized as follows. Section 2 proves the hardness results. Section 3 presents a PTAS for c-FRAGMENTED LAPCS(NESTED, NESTED), for any constant $c \geq 2$, and then extends it to a PTAS for c-DIAGONAL LAPCS(NESTED, NESTED), for any constant $c \geq 1$. Section 3 also presents an efficient exact algorithm for 1-FRAGMENTED LAPCS(CROSSING, CROSSING). Section 4 deals with an interesting special case called UNARY LAPCS(NESTED, NESTED), proves its NP-hardness, and presents a $\frac{4}{3}$-approximation algorithm. Section 5 concludes the paper with some future research topics. In this extended abstract, proofs of some lemmas and theorems are omitted and are available in the full paper [20] (`http://www.math.uwaterloo.ca/~ghlin`).

2 Hardness Results

We begin with reviewing hardness results for some graph-theoretic problems.

Lemma 1. *The* MAXIMUM INDEPENDENT SET (MIS) *problem restricted to cubic planar graphs is NP-hard* [13]*. On the other hand, it admits a PTAS* [3]*.*

A *book embedding* of a graph consists of an embedding of its vertices along the spine of a book (*i.e.* a linear ordering of the vertices), and an embedding of its edges on the pages so that edges embedded on the same page do not cross. The objective of the BOOK EMBEDDING PROBLEM is to minimize the number of pages used. The minimum number of pages in which a graph can be embedded is called the *pagenumber* of the graph. Computationally, the BOOK EMBEDDING

PROBLEM is hard: it is NP-complete to tell if a graph can be embedded in two pages [9]. The restricted problem of embedding the edges optimally for a fixed vertex-ordering is also NP-complete [12]. Nonetheless, we have the following positive results due to Yannakakis.

Lemma 2. [25] *Graphs with pagenumber one are exactly the outerplanar graphs. Graphs with pagenumber two are the subhamiltonian planar graphs, i.e. the subgraphs of Hamiltonian planar graphs. There exists an algorithm which embeds a given planar graph in four pages in time linear in the number of edges in the graph. Moreover, there exist planar graphs which cannot be embedded in three pages.*

From Lemmas 1 and 2, we can use the reduction technique in [11,17] to construct a simple reduction showing the NP-hardness of the LAPCS problem for four nested arc-annotated sequences (the definition is a straightforward extension of that of LAPCS(NESTED, NESTED)). In Theorem 1 below, we prove the hardness result for two nested arc-annotated sequences. We need the following improvements on Lemmas 1 and 2 to prove the theorem.

Lemma 3. [5] *The* VERTEX COVER (VC) *problem restricted to cubic triconnected planar graphs is NP-hard.*

Corollary 1. *The* MIS *problem restricted to cubic planar bridgeless connected graphs is NP-hard.*

Lemma 4. *Any cubic planar bridgeless graph has a perfect matching [21], which can be computed in linear time [4].*

The following lemma is crucial to the proof [20] of the theorem.

Lemma 5. [20] *Cubic planar bridgeless connected graphs are subhamiltonian. Moreover, there is a linear-time algorithm that, given any cubic planar bridgeless connected graph G, finds a Hamiltonian planar supergraph $H(G)$ of G with maximum degree at most 5, and finds a 2-page book embedding of G such that every vertex has a degree at least one (and thus at most two) in each page.*

Theorem 1. LAPCS(NESTED, NESTED) *is NP-hard.*

The reader may notice that the proof [20] of Theorem 1 can be made much simpler by employing a reduction similar to those in [11,17]. The reason why we prefer a more complicated reduction is that this reduction actually shows the NP-hardness of the problem of finding a *canonical* LAPCS for two given nested arc-annotated sequences. Here, in a canonical LAPCS, the allowed forms of base matches are $\langle 2i-1, 2i-1\rangle$, $\langle 2i-1, 2i\rangle$, $\langle 2i, 2i-1\rangle$, and $\langle 2i, 2i\rangle$. We use 2-FRAGMENTED LAPCS(NESTED, NESTED) to denote the problem of finding a canonical LAPCS for two given nested arc-annotated sequences. Generally, we can define c-FRAGMENTED LAPCS(NESTED, NESTED) for any positive integer c, in which the two given sequences are chopped into fragments of lengths at most c, and the allowed base matches are those between fragments at the same location. Notice that c-FRAGMENTED LAPCS(NESTED, NESTED) is actually a further restriction of $(c-1)$-DIAGONAL LAPCS(NESTED, NESTED).

Theorem 2. c-FRAGMENTED LAPCS(NESTED, NESTED) *is NP-hard when* $c \geq 2$. c-DIAGONAL LAPCS(NESTED, NESTED) *is NP-hard when* $c \geq 1$.

3 c-Fragmented LAPCS and c-Diagonal LAPCS

3.1 1-Fragmented LAPCS(Crossing, Crossing)

Let $(S_1, P_1), (S_2, P_2)$ be the two sequences in an instance of 1-fragmented LAPCS(crossing, crossing). Assume without loss of generality that $n = |S_1| = |S_2|$. Clearly, ignoring the arcs we may easily compute a classical LCS T of S_1 and S_2 by a linear scan in $O(n)$ time. We construct a graph G associated with T as follows. If T induces base match $\langle i, i\rangle$, then create a vertex v_i. If T induces a pair of base matches $\langle i, i\rangle$ and $\langle j, j\rangle$, and (i, j) is an arc in either P_1 or P_2 but not both, then we impose an edge connecting v_i and v_j in G. It is clear that G has maximum degree 2, and thus every independent set of G one-to-one corresponds to an arc-preserving common subsequence of (S_1, P_1) and (S_2, P_2). Therefore, by computing a maximum independent set of G, which can be easily done in linear time since G is composed of a collection of disjoint cycles and paths, we get an LAPCS for (S_1, P_1) and (S_2, P_2).

Theorem 3. *The* 1-fragmented LAPCS(crossing, crossing) *problem is solvable in* $O(n)$ *time.*

3.2 c-Fragmented LAPCS(Nested, Nested)

Before describing the PTAS, we review the notion of *tree decomposition* of a graph and *k-outerplanar graph*, which play important roles in our construction.

A *tree decomposition* of a graph $G = (V, E)$ is a pair $\mathcal{D} = (T, \mathcal{X})$ where $T = (U, F)$ is a tree and $\mathcal{X} = \{X_u \mid u \in U\}$ is a family of $|U|$ subsets of V such that the following hold:

- $\cup_{u \in U} X_u = V$.
- For each edge $(v_1, v_2) \in E$, there is a vertex $u \in U$ such that $\{v_1, v_2\} \subseteq X_u$.
- If $u_2 \in U$ is on the path connecting u_1 and u_3 in T, then $X_{u_1} \cap X_{u_3} \subseteq X_{u_2}$.

The *treewidth* associated with this decomposition is $tw(G, \mathcal{D}) = \max_{u \in U} |X_u| - 1$. The *treewidth* of G, denoted by $tw(G)$, is the minimum $tw(G, \mathcal{D})$ taken over all tree decompositions $\mathcal{D}$ of G.

The following lemma is widely known in the literature (see, *e.g.* [1]):

Lemma 6. *The* MIS *problem restricted to bounded-treewidth graphs is solvable in linear time.*

The notion of k-outerplanar graphs was introduced by Baker [3]. These graphs are defined inductively as follows. 1-*outerplanar graphs* are exactly outerplanar graphs. For $k \geq 2$, k-*outerplanar graphs* are those planar graphs that have a planar embedding such that deleting all vertices on the exterior face of the embedding and all edges incident to them from the embedding yields a $(k-1)$-outerplanar graph.

Lemma 7. [6] *For $k \geq 1$, k-outerplanar graphs have treewidth less than or equal to $3k - 1$.*

By Lemmas 6 and 7, the MIS problem restricted to k-outerplanar graphs is solvable in linear time. This result was originally due to Baker [3] but based on a different approach.

For any integer $k \geq 2$, a *k-cover* of a graph $G = (V, E)$ is a family $\mathcal{F}$ of k subsets of V such that each vertex of G appears in at least $k-1$ subsets in $\mathcal{F}$. The notion of k-covers of graphs was introduced by Chen [8]. The idea behind Baker's PTAS [3] for the MIS problem restricted to planar graphs is to compute a k-cover $\mathcal{F}$ of a given planar graph G such that each subset in $\mathcal{F}$ induces a $(k-1)$-outerplanar subgraph of G. Extending this idea and applying Lemma 6, Chen proved the following:

Lemma 8. [8] *Suppose that $\mathcal{C}$ is a class of graphs $G = (V, E)$ such that given G and an integer $k \geq 2$, we can compute a k-cover $\mathcal{F}$ of G in time polynomial in $k|V|$ such that each subset in $\mathcal{F}$ induces a subgraph of G whose treewidth is bounded from above by a number independent of G. Then, the* MIS *problem restricted to graphs in $\mathcal{C}$ admits a PTAS.*

Now we are ready to present the PTAS for c-FRAGMENTED LAPCS(NESTED, NESTED).

Theorem 4. *The c-*FRAGMENTED LAPCS(NESTED, NESTED) *problem admits a PTAS.*

Proof. Given an instance of c-FRAGMENTED LAPCS(NESTED, NESTED), say (S_1, P_1) and (S_2, P_2), and an integer k, we are asked to compute an arc-preserving common subsequence whose length is at least $\frac{k-1}{k}$ of the optimum. Without loss of generality, we assume that $|S_1| = |S_2| = mc\ (= n)$, and let S_i^j denote the jth fragment of length c in S_i. We construct two graphs G and H in the following.

Let V_j, $1 \leq j \leq m$, denote the set of vertices each corresponding to a base match $\langle (j-1)c + \ell_1, (j-1)c + \ell_2 \rangle$ in the jth fragment (*i.e.* $S_1[(j-1)c + \ell_1] = S_2[(j-1)c + \ell_2]$), where $1 \leq \ell_1, \ell_2 \leq c$. Clearly, $|V_j| \leq c^2$ for all j. Let $V = \cup_{1 \leq j \leq m} V_j$. Two vertices in V are connected via an edge if and only if they are conflicting base matches (*i.e.* they cross each other or violate the arc-preserving constraint). This forms a graph G. Observe that an independent set in G one-to-one corresponds to an arc-preserving common subsequence of (S_1, P_1) and (S_2, P_2) (with the c-fragmented restriction). So, by Lemma 8, it suffices to show that given G and an integer $k \geq 2$, we can compute a k-cover $\mathcal{F}$ of G in time polynomial in $k|V|$ such that each subset in $\mathcal{F}$ induces a subgraph of G whose treewidth is bounded from above by a number independent of G.

Consider an integer $k \geq 2$. We obtain a new graph H from G by merging the vertices in each subset V_j into a super-vertex v_j and keeping at most one edge between each pair of super-vertices. Since both P_1 and P_2 are nested, H is a simple planar graph. Notice that the number of vertices in H is $2m$.

Since H is planar, we can emulate Baker [3] to compute a k-cover $\mathcal{F}_H$ of H in time linear in $2mk$ such that each subset in $\mathcal{F}_H$ induces a $(k-1)$-outerplanar subgraph of G. Let $\mathcal{F}_H = \{U_1, \ldots, U_k\}$. For each $U_i \in \mathcal{F}_H$, we can compute a tree decomposition $\mathcal{D}_i = (T_i, \mathcal{X}_i)$ of the subgraph H_i of H induced by U_i with $tw(H_i, \mathcal{D}_i) \leq 3k - 4$, in time linear in $|U_i|$ [7].

For each $U_i \in \mathcal{F}_H$, let W_i be the subset of V obtained from U_i by replacing each super-vertex $v \in U_i$ with the original vertices merged into v. Similarly, for each $1 \leq i \leq k$ and each subset $X_{i,j} \in \mathcal{X}_i$, we obtain a subset $Z_{i,j}$ of V from $X_{i,j}$ by replacing each super-vertex $v \in X_{i,j}$ with the original vertices merged into v. It is easy to verify that (1) $\{W_1, \ldots, W_k\}$ is a k-cover of G, and (2) each $\mathcal{D}_i = (T_i, \{Z_{i,j} \mid X_{i,j} \in \mathcal{X}_i\})$ is a tree decomposition of the subgraph of G induced by W_i. Note that the treewidth of each $\mathcal{D}_i$ is less than or equal to $(3k-4)c^2$. This finishes the proof of the theorem. □

3.3 c-Diagonal LAPCS(Nested, Nested)

The PTAS in Section 3.2 can be easily extended to a PTAS for c-diagonal LAPCS(nested, nested).

Theorem 5. *The c-diagonal LAPCS(nested, nested) problem admits a PTAS.*

Proof. Given an instance of c-diagonal LAPCS(nested, nested), say (S_1, P_1) and (S_2, P_2), and an integer k, we are asked to compute an arc-preserving common subsequence whose length is at least $\frac{k-1}{k}$ of the optimum. Without loss of generality, we assume that $|S_1| = |S_2| = n$. The algorithm uses the PTAS designed for c-fragmented LAPCS(nested, nested) as a subroutine.

Fix a constant b (to be specified later). For every $i \in [1, b]$, let Q_i denote the b-fragmented LAPCS(nested, nested) problem for (S_1, P_1) and (S_2, P_2) where the first fragment has length i, each of the others but the last fragment has length b. Let ℓ_i^* denote the length of an LAPCS, say T_i, for problem Q_i. Suppose that R is an LAPCS for the original problem. Then, every base match in R is a legal base match in at least $(b-c+1)$ out of b problems Q_1, Q_2, $\cdots$, Q_b. Therefore, the maximum among ℓ_1^*, ℓ_2^*, $\cdots$, ℓ_b^* is at least as large as $\frac{b-c+1}{b}|R|$. Notice that by Theorem 4 we may compute an arc-preserving common subsequence C_i for problem Q_i in polynomial time such that the length of C_i is at least as large as $\frac{b}{b+1}\ell_i^*$.

Let C be the longest one among $C_1, \ldots, C_b$. By the discussions in the previous paragraph, $|C| \geq \frac{b}{b+1} \cdot \frac{b-c+1}{b}|R| = \frac{b-c+1}{b+1}|R|$. By setting $b = ck - 1$, we have $|C| \geq \frac{k-1}{k}|R|$. Thus, the algorithm that outputs C on input (S_1, P_1), (S_2, P_2) and an integer k is a PTAS for the c-diagonal LAPCS(nested, nested) problem. □

4 Unary LAPCS(Nested, Nested)

Unary LAPCS(nested, nested) is a special case where all the bases are the same. Although the problem has no direct applications in molecular biology, it arises in several other domains such as query optimization (parenthesis matching) and planning.

Theorem 6. *The* unary LAPCS(nested, nested) *problem is NP-hard.*

The following corollary follows from the construction in the proof [20].

Corollary 2. *c*-FRAGMENTED UNARY LAPCS(NESTED, NESTED) *when* $c \geq 4$ *and* *c*-DIAGONAL UNARY LAPCS(NESTED, NESTED) *when* $c \geq 3$ *are NP-hard.*

Theorem 7. UNARY LAPCS(NESTED, NESTED) *admits a* $\frac{4}{3}$*-approximation algorithm.*

Proof. Given a pair of unary sequences with nested arcs (S_1, P_1) and (S_2, P_2), where $|S_1| = n_1 \leq n_2 = |S_2|$, an arc-preserving common subsequence T satisfies *left-priority* if for every arc $(i_1, i_2) \in P_1$, T does not contain $S_1[i_2]$ unless it contains $S_1[i_1]$. For the ease of exposition, we call such an arc-preserving common subsequence satisfying left-priority a *Lep-APCS*. A longest Lep-APCS is denoted for short by *Lep-LAPCS*. A key ingredient of our approximation algorithm is a polynomial-time algorithm for computing a Lep-LAPCS, as shown below.

The basic idea is dynamic programming. For every pair of integers i and i', where $1 \leq i \leq i' \leq n_1$, let $\widetilde{S}_1[i, i']$ denote the subsequence of $S_1[i, i']$ by deleting the bases that are the right endpoints of arcs whose left endpoints are not in the interval $[i, i']$ [1], $\tilde{\imath}$ (or $\tilde{\imath}'$) the index of the base to the right (or left, respectively) of $S_1[i]$ (or $S_1[i']$, respectively) in $\widetilde{S}_1[i, i']$, and $P_1[i, i']$ the subset of arcs whose both endpoints are in the interval $[i, i']$.

Let $DP(i, i'; j, j')$ denote the length of a Lep-LAPCS for the subsequences $(\widetilde{S}_1[i, i'], P_1[i, i'])$ and $(S_2[j, j'], P_2[j, j'])$, where $1 \leq i \leq i' \leq n_1$ and $1 \leq j \leq j' \leq n_2$. Here we require that the pair j and j' satisfy that there is no arc in P_2 such that its one endpoint is in the interval $[j, j']$ while the other is not. The solution to our original Lep-LAPCS problem would be stored in $DP(1, n_1; 1, n_2)$. In the following, when we say that $S_1[k]$ (or $S_2[k]$) is *free*, it is not an endpoint of any arc in $P_1[i, i']$ (or $P_2[j, j']$, respectively). The dynamic programming is a two-step computation.

Step 1. For every $(j_1, j_2) \in P_2$, we do the following two-phase pre-computation.

Phase I: If $j_2 - j_1 > 1$, then let $j = j_1 + 1$. Let j' be a position in range $[j_1 + 1, j_2 - 1]$ such that $(j, j') \notin P_2$ and no arc in P_2 has exactly one endpoint in range $[j, j']$. The recurrence relation for the computation of entry $DP(i, i'; j, j')$ is defined for several cases.

If $S_2[j']$ is free, then in the case that $S_1[i']$ is free,

$$DP(i, i'; j, j') = \max \begin{cases} DP(i, i'; j, j' - 1), \\ DP(i, \tilde{\imath}'; j, j'), \\ DP(i, \tilde{\imath}'; j, j' - 1) + 1; \end{cases}$$

In the other case, *i.e.* $S_1[i']$ is the right endpoint of some arc in $P_1[i, i']$,

$$DP(i, i'; j, j') = \max \begin{cases} DP(i, i'; j, j' - 1), \\ DP(i, \tilde{\imath}'; j, j'). \end{cases}$$

[1] Note: We may assume without loss of generality that $S_1[i]$ and $S_1[i']$ are both in $\widetilde{S}_1[i, i']$. Otherwise, if, for example, $S_1[i]$ is not in $\widetilde{S}_1[i, i']$, then $\widetilde{S}_1[i, i'] = \widetilde{S}_1[i + 1, i']$.

If $(j'', j') \in P_2$ for some $j'' \in [j+1, j'-1]$, then

$$DP(i, i'; j, j') = \max_{i \le i'' \le i'} \{DP(i, i''-1; j, j''-1) + DP(i'', i'; j'', j')\}.$$

Phase II: If $S_1[i]$ is free but $S_1[i']$ isn't, then

$$DP(i, i'; j_1, j_2) = \max \begin{cases} DP(\tilde{i}, i'; j_1, j_2), \\ DP(\tilde{i}, i'; j_1+1, j_2-1)+1, \\ DP(i, \tilde{i}'; j_1, j_2). \end{cases}$$

Similarly, if $S_1[i']$ is free, then no matter whether or not $S_1[i]$ is free,

$$DP(i, i'; j_1, j_2) = \max \begin{cases} DP(\tilde{i}, i'; j_1, j_2), \\ DP(\tilde{i}, i'; j_1+1, j_2-1)+1, \\ DP(i, \tilde{i}'; j_1, j_2), \\ DP(i, \tilde{i}'; j_1+1, j_2-1)+1. \end{cases}$$

If $(i, i') \in P_1[i, i']$, then

$$DP(i, i'; j_1, j_2) = DP(\tilde{i}, \tilde{i}'; j_1+1, j_2-1)+2.$$

If neither of $S_1[i]$ and $S_1[i']$ is free, but $(i, i') \notin P_1[i, i']$, then

$$DP(i, i'; j_1, j_2) = \max \begin{cases} DP(i, i'; j_1+1, j_2-1), \\ DP(\tilde{i}, i'; j_1+1, j_2-1)+1, \\ DP(i, \tilde{i}'; j_1, j_2), \\ DP(\tilde{i}, i'; j_1, j_2). \end{cases}$$

Step 2. Let j' be a position in range $[1, n_2]$ such that $(1, j') \notin P_2$ and no arc in P_2 has exactly one endpoint in range $[1, j']$. The recurrence relation for the computation of entry $DP(i, i'; 1, j')$ is defined for several cases.

If $S_2[j']$ is free, then in the case that $S_1[i']$ is free,

$$DP(i, i'; 1, j') = \max \begin{cases} DP(i, i'; 1, j'-1), \\ DP(i, \tilde{i}'; 1, j'), \\ DP(i, \tilde{i}'; 1, j'-1)+1; \end{cases}$$

In the other case, *i.e.* $S_1[i']$ is the right endpoint of some arc in $P_1[i, i']$,

$$DP(i, i'; 1, j') = \max \begin{cases} DP(i, i'; 1, j'-1), \\ DP(i, \tilde{i}'; 1, j'). \end{cases}$$

If $(j'', j') \in P_2$ for some $j'' \in [2, j'-1]$, then

$$DP(i, i'; 1, j') = \max_{i \le i'' \le i'} \{DP(i, i''-1; 1, j''-1) + DP(i'', i'; j'', j')\}.$$

Therefore, $DP(1, n_1; 1, n_2)$ can be computed in $O(n_1^3 n_2)$ time, since there are $O(n_1^2 n_2)$ entries and each entry is obtained by checking at most $O(n_1)$ terms.

Employing a standard back-tracing technique, a Lep-LAPCS can be recovered in $O(n_1 n_2)$ time. Let n_1^* denote the length of a Lep-LAPCS.

We can similarly define arc-preserving common subsequences satisfying *right-priority*, and denote the longest ones as *Rip-LAPCS*'s. Using the same computation technique as in the above, a Rip-LAPCS and its length can be computed in $O(n_1^3 n_2)$ time too. Let n_2^* denote the length of a Rip-LAPCS.

Let T_0 denote a longest APCS under the constraint that it contains no arc-match at all, and $n_0^* = |T_0|$. Such a T_0 can be easily computed by deleting all bases which are right endpoints of arcs in P_1 and P_2 from both sequences and then taking the shorter one. Let n^* denote the length of a (genuine) LAPCS T^* for (S_1, P_1) and (S_2, P_2), and m^* denote the number of arc-matches in T^*. If $m^* \leq \frac{1}{4}n^*$, then we conclude that $n_0^* \geq n^* - m^* \geq \frac{3}{4}n^*$. Otherwise, the number of base-matches in T^* which do not form arc-matches is less than $\frac{1}{2}n^*$. Therefore, $\max\{n_1^*, n_2^*\} > \frac{3}{4}n^*$. It follows that by taking the longest one among T_0, a Lep-LAPCS, and a Rip-LAPCS, as the approximate solution, its length is guaranteed to be at least $\frac{3}{4}$ of the optimum. Notice that the overall algorithm runs in $O(n_1^3 n_2)$ time. □

5 Concluding Remarks

We have completely answered an open question proposed in [11] on whether or not it is NP-hard to compute an LAPCS for two nested arc-annotated sequences. However, does it admit a PTAS? We leave it as an open question. Designing better approximations for UNARY LAPCS(NESTED, NESTED) and/or LAPCS(CROSSING CROSSING) constitutes another challenging problem.

References

1. S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability: a survey. *BIT*, 25:2–23, 1985.
2. V. Bafna, S. Muthukrishnan, and R. Ravi. Computing similarity between RNA strings. In *Proceedings of 6th Annual Symposium on Combinatorial Pattern Matching (CPM'95)*, LNCS 937, pages 1–16, 1995.
3. B.S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41:153–180, 1994.
4. T.C. Biedl, P. Bose, E.D. Demaine, and A. Lubiw. Efficient algorithms for Peterson's matching theorem. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*, pages 130–139, 1999.
5. T.C. Biedl, G. Kant, and M. Kaufmann. On triangulating planar graphs under the four-connectivity constraint. *Algorithmica*, 19:427–446, 1997.
6. H.L. Bodlaender. Planar graphs with bounded treewidth. Technical Report RUU-CS-88-14, Department of Computer Science, Utrecht University, The Netherlands, March 1988.
7. H.L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.
8. Z.-Z. Chen. Efficient approximation schemes for maximization problems on $K_{3,3}$-free or K_5-free graphs. *Journal of Algorithms*, 26:166–187, 1998.

9. F.R.K. Chung, F.T. Leighton, and A.L. Rosenberg. Embedding graphs in books: a graph layout problem with applications to VLSI design. *SIAM Journal on Algebraic and Discrete Methods*, 8:33–58, 1987.
10. F. Corpet and B. Michot. RNAling program: alignment of RNA sequences using both primary and secondary structures. *Computer Applications in the Bio-sciences*, 10:389–399, 1994.
11. P.A. Evans. *Algorithms and Complexity for Annotated Sequence Analysis*. PhD thesis, University of Victoria, 1999.
12. M.R. Garey, D.S. Johnson, G.L. Miller, and C.H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic and Discrete Methods*, 1:216–227, 1980.
13. M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.
14. D. Goldman, S. Istrail, and C.H. Papadimitriou. Algorithmic aspects of protein structure similarity. In *IEEE Proceedings of the 40th Annual Conference of Foundations of Computer Science (FOCS'99)*, pages 512–521, 1999.
15. D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge, 1997.
16. D.S. Hirschberg. *The Longest Common Subsequence Problem*. PhD thesis, Princeton University, 1975.
17. T. Jiang, G.-H. Lin, B. Ma, and K. Zhang. The longest common subsequence problem for arc-annotated sequences. In *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching (CPM 2000)*, LNCS 1848, pages 154–165, 2000. Full paper accepted by *Journal of Discrete Algorithms*.
18. H. Lenhof, K. Reinert, and M. Vingron. A polyhedral approach to RNA sequence structure alignment. In *Proceedings of the Second Annual International Conference on Computational Molecular Biology (RECOMB'98)*, pages 153–159, 1998.
19. M. Li, B. Ma, and L. Wang. Near optimal multiple sequence alignment within a band in polynomial time. In *ACM Proceedings of the 32nd Annual Symposium on Theory of Computing (STOC'00)*, pages 425–434, 2000.
20. G.-H. Lin, Z.-Z. Chen, T. Jiang, and J.-J. Wen. The longest common subsequence problem for sequences with nested arc annotations, February 2001. Manuscript.
21. J. Peterson. Die theorie der regulären graphs (the theory of regular graphs). *Acta Mathematica*, 15:193–220, 1891.
22. D. Sankoff. Simultaneous solution of the RNA folding, alignment, and protosequence problems. *SIAM Journal on Applied Mathematics*, 45:810–825, 1985.
23. T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.
24. R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21:168–173, 1974.
25. M. Yannakakis. Embedding planar graphs in four pages. *Journal of Computer and System Sciences*, 38:36–67, 1989.
26. K. Zhang, L. Wang, and B. Ma. Computing similarity between RNA structures. In *Proceedings of 10th Annual Symposium on Combinatorial Pattern Matching (CPM'99)*, LNCS 1645, pages 281–293, 1999.

Visibility-Based Pursuit-Evasion in a Polygonal Region by a Searcher*

Sang-Min Park[1], Jae-Ha Lee[2], and Kyung-Yong Chwa[1]

[1] Dept. of Computer Science, KAIST, Korea
{smpark,kychwa}@jupiter.kaist.ac.kr
[2] Max-Planck-Institut für Informatik, Germany
lee@mpi-sb.mpg.de

Abstract. We consider the most basic visibility-based pursuit-evasion problem defined as follows: Given a polygonal region, a searcher with 360° vision, and an *unpredictable* intruder that is arbitrarily faster than the searcher, plan the motion of the searcher so as to *see* the intruder. In this paper, we present simple necessary and sufficient conditions for a polygon to be searchable, which settles a decade-old open problem raised in [13]. We also show that every searchable polygon is also searchable by a searcher with two flashlights (that is, two ray visions). This implies, combined with the previous work [7], that there is an $O(n^2)$-time algorithm for constructing a search path for an n-sided polygon.

1 Introduction

Background. The visibility-based pursuit-evasion problem is that of planning the motion of one or more searchers in a polygonal environment to eventually see an intruder that is unpredictable, has unknown initial position, and is capable of moving arbitrarily fast. This problem can model many practical applications such as search for an intruder in a house, rescue of a victim in a dangerous house and other surveillance with autonomous mobile robots. The motion plan calculated could be used by robots or human searchers.

This paper discusses the most basic problem with a single searcher. Imagine that a detective has to find a fugitive in a house with no way out. Probably, she should drive the fugitive into a corner by looking at every suspicious corner one by one. Can she eventually see the fugitive who is much faster? Or can the fugitive keep sneaking out of sight of the detective? It depends on the geometry of the house. More formally, we assume that the searcher's environment is a simple polygon. The searcher is equipped with the 360° vision that provides the visibility polygon in real time (also called ∞-searcher in [13,1]). The intruder is assumed to have an unknown initial position and to be capable of moving arbitrarily fast, whereas the searcher can move at bounded speed. The question is whether the searcher can eventually see the intruder controlled by the adversary. If this is the case, the polygon is said to be *searchable*.

* This work was supported by KOSEF(Korea Science and Engineering Foundation) under grant 98-0102-0701-3.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 456–468, 2001.

Related Work. There have been many studies on detecting the unpredictable intruder. The general problem was first studied in the context of graphs, where the searchers and the intruder can move from vertex to vertex until a searcher and the intruder eventually lie in one vertex [11,9]. After adopting geometric free-space constraints and visibility of the searchers, this problem has attracted much attention in computational geometry and robotics [13,3,1,2,15,6,8,4].

As the first attempt, Suzuki and Yamashita [13] introduced the polygon search problem, which is the topic of this paper. They presented some necessary or sufficient conditions for a polygon to be searchable by a single searcher but no complete characterizations. Guibas *et al.* [2] presented a complete algorithm to search a polygon by a single searcher, but the complexity of this problem, including the analysis of their algorithm, remains open. Recently, there was an erroneous result on the polygon search problem [14], which turned out to consider the restricted searcher [10].

As a characterizable variant of the polygon, many researchers considered *rooms* and *corridors*. A room is a polygonal region having one door and the searcher should detect the intruder while not allowing the intruder to reach the door. Lee *et al.* [8] characterized the class of searchable rooms and presented an $O(n^2)$-time algorithm for constructing a search path. A corridor is a polygonal region having two doors and should be searched from one door d to the other door d' in such a way that an intruder should be detected or evicted through d'. Crass *et al.* [1] characterized the class of searchable corridors. The characterizations for rooms and corridors rely on the *'orientation'* implicitly defined by the doors and thus cannot be applied to the polygon search problem.

Suzuki and Yamashita [13] defined searchers with various visibilities. The *k-searcher* has k flashlights each of which provides the scene along one ray that can be rotated at some bounded speed. It has been conjectured that any searchable polygon is also searchable by a 2-searcher, which is the motivation of the introduction of the k-searcher. This conjecture is important from the algorithmic point of view because the authors of the present paper gave an $O(n^2)$-time algorithm for constructing a search schedule of the 2-searcher [7], which is also complete for the ∞-searcher if the above conjecture is true.

Our Results. This paper solves the polygon search problem. We first give three necessary conditions, say N1, N2, and N3, for a polygon to be searchable and show that they are sufficient. As a by-product, we show that any searchable polygon is also searchable by a 2-searcher, which means that the $O(n^2)$-time algorithm for the 2-searcher [7] is also complete for the general searcher. Due to the lack of space, we sometimes omit the detailed proof, which can be found in the full paper [10].

2 Preliminaries

2.1 Definitions and Notations

We are given a simple polygon P in the plane. The searcher and the intruder are modeled as points that move continuously within P. Let $\phi(t)$ denote the

position of the intruder at time $t \geq 0$. It is assumed that $\phi : [0, \infty) \to P$ is a continuous function, and the intruder is unpredictable in that he is capable of moving arbitrarily fast and his path ϕ is unknown to the searcher.

Let $\gamma(t)$ denote the position of the searcher at time $t \geq 0$. Let γ represent a continuous path of the searcher of the form $\gamma : [0, \infty) \to P$. The initial position of the searcher has no importance because she can move to any point and start searching there. Two points p and q are *visible* from each other if the segment $\overline{pq}$ does not intersect the exterior of P. For any point $q \in P$, let $Vis(q)$ denote the set of all points in P that are visible from q. The searcher has the 360° vision, i.e., she sees all the points in $Vis(\gamma(t))$ at t. The polygon P is *searchable* if there exists a path γ such that for every continuous function $\phi : [0, \infty) \to P$, there is a time instant $t \in [0, \infty)$ such that $\phi(t) \in Vis(\gamma(t))$. That is, the intruder will be seen by the searcher regardless of his path ϕ. Figure 1 depicts an example of a searchable polygon and its search path.

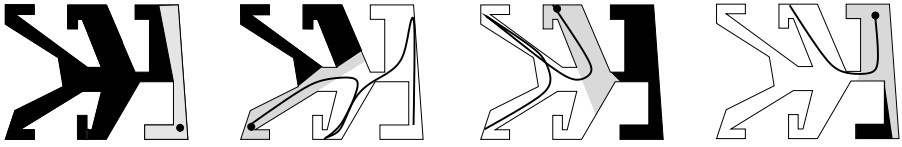


Fig. 1. Some snapshots of a search path of the searcher. Gray region is currently visible from the searcher and dark region might contain the intruder.

Let ∂P denote the boundary of P. Since the intruder has unbounded speed, any path of the undetected intruder can be mapped to a path of the undetected intruder along ∂P. Thus, we assume w.l.o.g. that the intruder always moves along the boundary of P in the rest of this paper. It is assumed that ∂P is oriented in the *clockwise* direction. For points $p, q \in \partial P$, the *chain* $[p, q]$ denotes the connected boundary chain from p to q in the clockwise direction. Analogously, (p, q) denotes $[p, q] \setminus \{p, q\}$, $(p, q]$ does $[p, q] \setminus \{p\}$, and $[p, q)$ does $[p, q] \setminus \{q\}$.

An edge e of P is contained in a line l; an *extended edge* is a maximal line segment $e' \subset (l \cap P)$ such that it shares only one endpoint (but no interior point) with e and its other endpoint also lies on ∂P. Each edge induces at most two extended edges. We call an extended edge e' a *cut* and we call e the *inner edge* of the cut e' (see Figure 2a). Every cut e' divides the polygon into two parts: one containing e, called the *inside* of e', and the other. Let us denote two endpoints of a cut e' by $B(e')$ and $U(e')$ such that the inside of e' is bounded by e' and the chain $[B(e'), U(e')]$. A chain $\mathcal{C}$ is said to *contain* a cut e', if $[B(e'), U(e')] \subseteq \mathcal{C}$. A cut e' is an *essential cut* if $[B(e'), U(e')]$ contains no other cuts (see Figure 2a).

For a region R in P, the *visibility polygon of* R, written $Vis(R)$, specifies the set of all points in P each of which is visible from some point in R. See Figure 2b. We define a *cave of a cut* e' to be a (open) connected chain of ∂P whose two endpoints lie in $Vis(e')$ but no other points. A cave of e' is called a *left cave* if it lies to the left of the boundary of $Vis(e')$; otherwise, it is called a *right cave*.

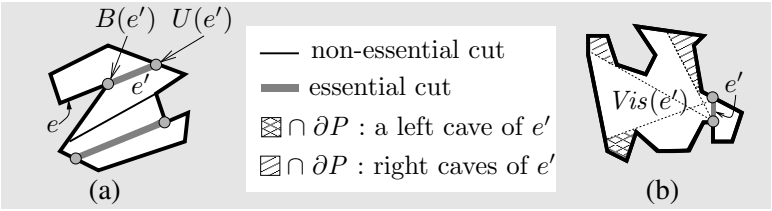


Fig. 2. Definition of cuts, essential cuts and caves.

2.2 Clear vs. Contaminated

Any region that might contain an undetected intruder is said to be *contaminated*; otherwise, it is said to be *clear*. And a cut c is defined to be *clear* if the inner edge of c is clear; otherwise it is *contaminated*. Initially, the whole polygon P is contaminated and the searcher should clear P. The searcher crosses or touches every cut, because to clear the inner edge of a cut, the searcher must visit the inside of the cut. We assume that P has at least two essential cuts; otherwise the searching is trivial. Actually, if you visited two essential cuts a and b, you must have visited all the cuts containing one of a and b. Thus, we will concentrate on essential cuts only.

Suppose that γ is a search path for P. Consider a fixed time t. If a region is contaminated right before t and is clear at t, it is said to *be cleared* or *becomes clear* at t. The following observations are easily seen from the assumption that the intruder can move arbitrarily fast.

Fact 1 *Any region that is disjoint from $Vis(\gamma(t))$ is entirely clear or entirely contaminated at t.*

Fact 2 *If a cut c is cleared at t, $\gamma(t)$ lies in the inside of c.*

However, simply visiting all the essential cuts does not make a successful search. Sometimes the searcher should clear a particular region again, if the intruder sneaked into the region; we say that the region is *recontaminated*. To deal with the recontamination, we focus on the cuts that are unable to prevent the neighboring cuts being recontaminated. Let $c_0, c_1, \cdots, c_m (= c_0)$ denote essential cuts ordered by $B(c_i)$ in the clockwise direction. If a cave of c_i contains c_{i+1}, we call it the *cw*-cave of c_i. Symmetrically, if a cave of c_i contains c_{i-1}, we call it the *ccw*-cave of c_i. Notice that a cave of c_i may contain both c_{i-1} and c_{i+1} (for example, see cut a in Figure 3a); in this case, the *cw*-cave of c_i is the same as the *ccw*-cave of c_i and we say that c_i is a *deep* cut. Observe that the *cw*-cave of c_i may be a left or a right cave of c_i (for example, see cut a in Figure 3a). Let $\mathcal{N}_L$ be the set of cuts that have a *cw*-cave and $\mathcal{N}_R$ be the set of cuts with a *ccw*-cave. We denote the union of $\mathcal{N}_L$ and $\mathcal{N}_R$ by $\mathcal{N}$.

If a cut c_i has both a *cw*-cave and a *ccw*-cave which are different (i.e., c_i is not deep), then we duplicate c_i, in $\mathcal{N}$, into imaginary twin cuts, say c_i^l and c_i^r, so that each copy has the unique *cw*-cave or *ccw*-cave but not both. For a

cut $c_i \in \mathcal{N}_L$(or $\mathcal{N}_R$), we denote the *cw*-cave (or *ccw*-cave) of c_i by $Q(c_i)$ and its complement by $D(c_i)$. That is, $D(c_i) = \partial P \setminus Q(c_i)$. (Observe that points in $D(c_i)$ are not necessarily visible from c_i.) Two cuts c_i and c_j are said to be *non-dominating each other* if c_i is contained in $Q(c_j)$ and c_j is contained in $Q(c_i)$. Then the following is easily shown.

Lemma 1 *Let a and b be cuts that are non-dominating each other. If $\{a\} \cup \{b\}$ becomes clear at t, either $Q(a)$ or $Q(b)$ is clear right before t.*

3 Necessary Conditions for Searchable Polygons

In this section we present three necessary conditions for a polygon to be searchable. Sufficiency of these conditions will be shown in Section 4.

u-triple. Let us begin with examples that are known to be not searchable. See Figure 3a. We call them an *unguarded triple*, shortly *u-triple*. Formally, three cuts a, b, and c form a u-triple, if any two of them are contained in one cave of the other. The following theorem is not new (refer to [13]).

Theorem 2 (N1) *If a polygon contains a u-triple, it is not searchable.*

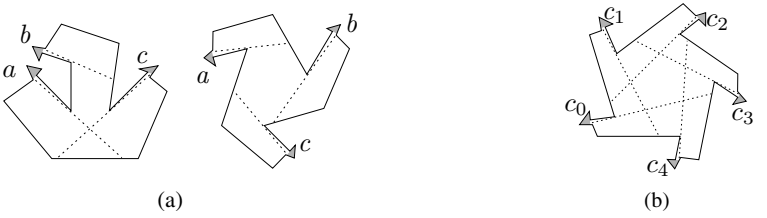


Fig. 3. Examples violating N1(a) and N2(b) :shaded regions denote the inside of cuts.

2-cover. Another example that is known to be not searchable consists of five cuts (Figure 3b). This is called *5-windmill*. Interestingly, we can extend this forbidden pattern into k-windmill using k cuts in $\mathcal{N}_L$(or $\mathcal{N}_R$) for any $k \geq 5$. We introduce a new concept of *2-cover* that explains why k-windmills are not searchable. A set $X \subseteq \partial P$ *covers* a cut c if $[B(c), U(c)] \cap X$ is non-empty.

Definition 1 For a subset S of $\mathcal{N}_L$(or $\mathcal{N}_R$), two essential cuts a and b form a *2-cover of S* if $a, b \in S$ and $D(a) \cup D(b)$ covers every cut in S.

Theorem 3 (N2) *If a polygon is searchable, there exists a 2-cover of S for any set S that is a subset of $\mathcal{N}_L$ or $\mathcal{N}_R$.*

Sketch of Proof: An example violating N2 is given in Figure 3b. The set $\{c_i \in \mathcal{N}_L \mid 0 \leq i \leq 4\}$ has no 2-cover.

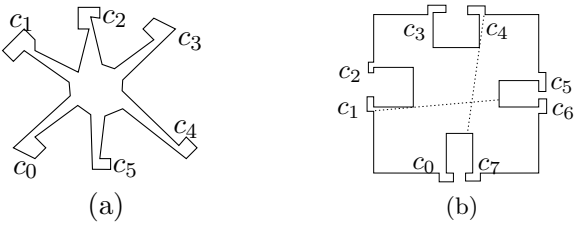


Fig. 4. Examples of non-searchable polygons that violate N3.

For contradiction, suppose that a set S has no 2-cover and that the polygon P is searchable. We assume w.l.o.g. that S is a subset of $\mathcal{N}_L$. We fix a search path γ for P. Let $c \in S$ denote the cut whose $Q(c)$ becomes clear first among the cuts in S by γ. Let t denote that time instant.

We claim that $Q(c)$ contains two cuts in S that are non-dominating each other. Let $Q(c) = (p, q)$. To show this claim, we first pick a cut in $S \cap Q(c)$ that is first met when we traverse ∂P from q counterclockwise; let a denote that cut. Since S has no 2-cover, at least one cut of S is contained in $Q(c) \cap Q(a)$. Let b be the cut of S in $Q(c) \cap Q(a)$ which is met first when we traverse ∂P from $B(a)$ counterclockwise. It is easily checked that a and b are non-dominating each other.

Thus $\{a\} \cup \{b\}$ is clear at t and there is a time instant t' $(\leq t)$ at which $\{a\} \cup \{b\}$ becomes clear. Since a and b are non-dominating each other, either $Q(a)$ or $Q(b)$ is clear right before t' by Lemma 1, which is a contradiction to the fact that $Q(c)$ becomes clear first among the cuts in S. □

s-triple. The third condition generalizes the idea used in the last necessary condition for a room to be searchable, described in [8].

Let $\mathcal{S} = \langle c_1, c_2, c_3 \rangle$ be a triple of essential cuts that appear along ∂P in this order. We call $\mathcal{S}$ an *s-triple* if c_1 has the *cw*-cave that is a left cave and both c_2 and c_3 lie in it, and c_3 has the *ccw*-cave that is a right cave and both c_1 and c_2 lie in it (in Figure 4b, $\langle c_1, c_3, c_4 \rangle$ is an s-triple). This definition is a slight modification of the s-triple defined in [8] and the proof of the following lemma is exactly same as that of Lemma 1 in [8].

Lemma 4 *Suppose that a polygon P is searchable and that γ is its search path. If $\langle c_1, c_2, c_3 \rangle$ is an s-triple, γ clears c_1 or c_3 last among $\{c_1, c_2, c_3\}$.*

If $\langle c_1, c_2, c_3 \rangle$ is an s-triple, we say that c_2 *has an s-pair* $\langle c_1, c_3 \rangle$. Lemma 4 says that any cut having an s-pair is not cleared last. The third necessary condition is a direct extension of this statement.

Theorem 5 (N3) *If every cut in P has an s-pair, P is not searchable.*

Examples that violate N3 are given in Figure 4. In Figure 4a, each c_i has an s-pair $\langle c_{i-1}, c_{i+1} \rangle$. In Figure 4b, both c_{2i} and c_{2i+1} have an s-pair $\langle c_{2i-1}, c_{2i+2} \rangle$.

4 Sufficiency

Theorem 6 *A polygon P is searchable, if it satisfies N1, N2, and N3.*

Throughout this section, we prove Theorem 6. To show that the polygon P is searchable, we consider a searcher with restricted visibility, called a *2-searcher* [13]. The 2-searcher has two flashlights whose visibility is limited to two rays emanating from her position, where the direction of each ray can be changed independently and continuously with bounded angular rotation speed. We will show that the polygon is 2-searchable if it satisfies three necessary conditions. This implies that the 2-searcher has the same search capability as the searcher with 360° vision, because any 2-searchable polygon is searchable. Actually, the 2-searcher has been used in the sufficiency proof of all previously-known characterizations [1,8].

The basic idea is to divide the search into three steps and to use a greedy algorithm within each step.

- FIRST STEP: The 2-searcher clears $D(c)$ for some $c \in \mathcal{N}$.
- SECOND STEP: The 2-searcher clears $Q(c)$. During this step, we allow the points in $D(c)$ to be recontaminated.
- THIRD STEP: If $D(c)$ was contaminated in the SECOND STEP, the 2-searcher clears it.

For example see Figure 1, where c is the lower right cut. The searcher clears $D(c)$ in (a), clears $Q(c)$ in (b), (c), and (d); during this SECOND STEP, $D(c)$ is recontaminated, so the searcher must visit c again in the THIRD STEP.

4.1 Movements of the 2-Searcher

Let us denote two flashlights of the 2-searcher by F_L and F_R. Suppose that the searcher stands at a point $s \in P$, aiming F_L and F_R at points p and q respectively (see Figure 5). Intuitively, we view the polygonal chain $\overline{p,s,q}$ as a variable-length two-segment chain $\mathcal{V}$ and search P by sweeping it with $\mathcal{V}$, satisfying the following invariant: *when $\mathcal{V}$ is $\overline{p,s,q}$, the chain $[q,p]$ is clear.* Especially, during the search, we keep $\mathcal{V}$ straight as long as possible. (Details about when we bend $\mathcal{V}$ are explained later.) While $\mathcal{V}$ is straight, we use a shorthand $\overline{p,q}$ to denote $\overline{p,s,q}$.

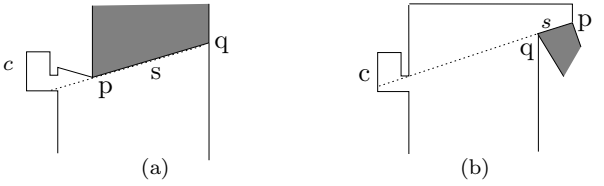


Fig. 5. Currently, $\mathcal{V}$ is at $\overline{p,s,q}$. The chain $[q,p]$ is clear.

We will use the sub-movements defined in [1] to advance $\mathcal{V}$. See Figure 6. Suppose that the current $\mathcal{V}$ is $\overline{p,q}$. If some chain $[p,r]$ is entirely visible from q, we can advance the left endpoint of $\mathcal{V}$ continuously to r, fixing the right endpoint (Figure 6a); we call this movement *l-advance-by-sweep*. If the clockwise neighborhood of p along ∂P is not visible from q, we take the point $r \in (p,q)$, closest to p, that is visible from q. Note that r lies on the line extending the segment $\overline{pq}$. If $[p,r]$ does not contain any essential cut, we can move $\mathcal{V}$ from $\overline{p,q}$ to $\overline{r,q}$ by performing *l-advance-along-lid* (Figure 6b and c), in which the searcher first moves to p fixing F_R at q and then clears $[p,r]$ by moving along the segment $\overline{pr}$ and by scanning the chain $[p,r]$ with F_L. (Note that $\mathcal{V}$ bends in this case.) Sometimes we can advance $\mathcal{V}$ by *l-advance-by-rotation* (Figure 6d), which is the 'rotation' from $\overline{p,q}$ to $\overline{a,b}$ centered at r' where $a \in (p,q)$ and $b \in (q,p)$ and $[p,a]$ is entirely visible from r'. Although the right endpoint of $\mathcal{V}$ may jump backwards in *l-advance-by-rotation*, no intruder can sneak into the region below $\mathcal{V}$. Actually, during the execution of all sub-movements no intruder can sneak into the region below $\mathcal{V}$. The *r-advance-by-sweep*,*r-advance-along-lid*, and *r-advance-by-rotation* are defined symmetrically. It will turn out that the above six types of sub-movements suffice to search any searchable polygon. Proofs of all previously-known characterizations used these sub-movements only [1,8].

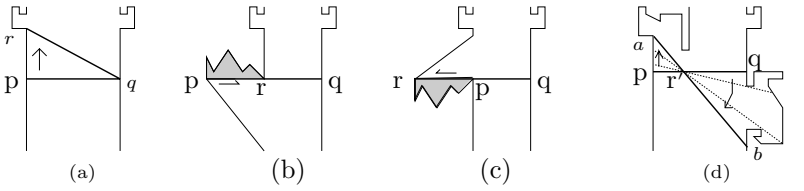


Fig. 6. The left endpoint of $\mathcal{V}$ is advancing.

Lemma 7 *If the 2-searcher moves from $\overline{p,s,q}$ to $\overline{p',s',q'}$ using sub-movements only, the 2-searcher can move from $\overline{q',s',p'}$ to $\overline{q,s,p}$ using sub-movements only.*

By this lemma, the THIRD STEP can be successfully done by the reverse execution of the FIRST STEP, if the latter is successfully done.

4.2 A Characterization of Searchable Corridors

To simplify the proof, we utilize a previously-known characterization of the class of searchable corridors. A corridor is a polygon with two edge doors, $d = \overline{pq}$ and $d' = \overline{p'q'}$. A corridor is *searchable* if the searcher starting at d can find the intruder or evict him through d' out of P in such a way that the intruder couldn't make a dash for the door d. Crass *et al.* [1] presented a characterization of searchable corridors and showed that the 2-searcher can search any searchable corridor, using sub-movements only. We restate it here in our notation.

Theorem 8 [1] *A corridor (P, d, d') for $d = \overline{pq}$ and $d' = \overline{p'q'}$ is searchable if and only if the following three conditions C1, C2, and C3 hold:*
(C1) *Every cut in $[p, p']$ is visible from some point in $[q', q]$; and every cut in $[q', q]$ is visible from some point in $[p, p']$.*
(C2) *There are no cuts a and b such that (i) a is contained in $[p, p']$ and b is contained in $[q', q]$, (ii) a has the cw-cave that is a left cave and that contains b and d', and (iii) b has the ccw-cave that is a right cave and that contains a and d'.*
(C3) *There are no cuts a and b such that (i) a is contained in $[p, p']$ and b is contained in $[q', q]$, (ii) a has the ccw-cave that is a right cave and that contains b and some point of d, and (iii) b has the cw-cave that is a left cave and that contains a and some point of d.*

A pair of cuts that violates C2 or C3 is called *"deadlock"*.

4.3 Second Step

The starting configuration of the SECOND STEP would be as in Figure 5: $D(c)$ is clear, $Q(c) = (p, q)$ is the *cw*-cave of c and contaminated, and $\mathcal{V}$ is $\overline{p, q}$. We will call such a configuration $Room(c, p, q)$. (In this paper, we mostly consider a *cw*-cave as a *Room*. This choice is arbitrary and symmetric arguments apply equally well to the case that a *Room* is a *ccw*-cave.)

The next lemma says that $\mathcal{V}$ can advance *locally*.

Lemma 9 *Suppose that the current configuration is $Room(c, p, q)$. If N1 holds, the 2-searcher either can clear the polygon or arrive at the configuration $Room(u, \cdot, \cdot)$ for the most counterclockwise cut u of $\mathcal{N}$ in $Q(c)$ such that q lies in the cw-cave of u.*

Sketch of Proof: Suppose the current configuration is $Room(c, p, q)$. See Figure 5. Thus $\mathcal{V}$ is $\overline{p, q}$. Let u be the most counterclockwise cut of $\mathcal{N}$ contained in $Q(c)$ such that q lies in the *cw*-cave of u. If such a cut u does not exist, the 2-searcher can clear $Q(c)$ using sub-movements.

Let $c_1 = c$, $\alpha(c_1) = q$, $\bar{\alpha}(c_1)$ be some point in $[B(c_1), U(c_1)]$ that is visible from q. We define c_i, $\alpha(c_i)$ and $\bar{\alpha}(c_i)$ for $i \geq 2$ inductively: c_i is the cut of $\mathcal{N}$ first met when we traverse ∂P from $B(c_{i-1})$ clockwise such that $\alpha(c_{i-1})$ lies in the *cw*-cave or the *ccw*-cave of c_i. If $\alpha(c_{i-1})$ is contained in the *cw*-cave (resp. *ccw*-cave) of c_i, $\alpha(c_i)$ is set to the point that is visible from some boundary point, say $\bar{\alpha}(c_i)$, in the inside of c_i and that is first met when we traverse ∂P from $\alpha(c_{i-1})$ clockwise (resp. counterclockwise). $\mathcal{V}$ advances step by step following c_i's. Suppose that $\mathcal{V}$ is currently at $\overline{\bar{\alpha}(c_{i-1}), \alpha(c_{i-1})}$. There are two cases (see Figure 7):

- Case 1. $\alpha(c_{i-1})$ lies in the *cw*-cave (say $\mathcal{S}$) of c_i. Assume that $\mathcal{S}$ is a left cave of c_i. See Figure 7a. Let $d_1(= c_{i-1}), d_2, \cdots, d_j(= c_i)$ be the essential cuts in this order. Let $\beta(d_k)$ denote the point that is visible from some boundary point in the inside of d_k, say $\bar{\beta}(d_k)$, and that is first met when we traverse ∂P from $\alpha(c_{i-1})$ clockwise. Then $\mathcal{V}$ can go through $\overline{\bar{\beta}(d_k), \beta(d_k)}$, for all $1 \leq k \leq j$,

using sub-movements. (In Figure 7a, $\mathcal{V}$ advances using *l-advance-by-sweep* and *l-advance-by-rotation* and dark regions are cleared by *l-advance-along-lid*.) If $\mathcal{S}$ is a right cave of c_i, we can show that no cut is contained in $[\alpha(c_i), \bar{\alpha}(c_i)]$. Thus the 2-searcher can easily move to $\overline{\bar{\alpha}(c_i), \alpha(c_i)}$.

– Case 2. $\alpha(c_{i-1})$ lies in the *ccw*-cave of c_i. See Figure 7b. In this case we can find a cut $c_j \in \mathcal{N}$ with the largest index j $(< i)$ such that $\alpha(c_i)$ lies in the *cw*-cave of c_j. Let us temporarily view $[\bar{\alpha}(c_j), \bar{\alpha}(c_i)] \cup [\alpha(c_i), \alpha(c_j)]$ as a corridor, where two doors are line segments $\overline{\bar{\alpha}(c_j)\alpha(c_j)}$ and $\overline{\bar{\alpha}(c_i)\alpha(c_i)}$. It is easily seen that this corridor satisfies C1 (that is, every cut in $[\bar{\alpha}(c_j), \bar{\alpha}(c_i)]$ is visible from some point in $[\alpha(c_i), \alpha(c_j)]$ and vice versa) because otherwise we can find a u-triple. Moreover it is easily seen that "deadlock" does not occur (otherwise, we can find a u-triple again). This means $[\bar{\alpha}(c_j), \bar{\alpha}(c_i)] \cup [\alpha(c_i), \alpha(c_j)]$ is a searchable corridor and $\mathcal{V}$ can advance to $\overline{\bar{\alpha}(c_i), \alpha(c_i)}$ using the algorithm described in [3].

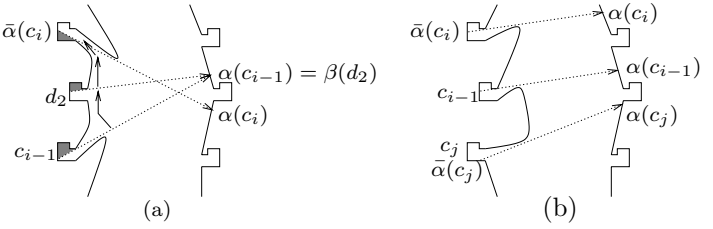


Fig. 7. Proof of Lemma 9. (a) Case 1. (2) Case 2.

In this way, the 2-searcher can advance step by step. Thus, either the 2-searcher can complete searching or the 2-searcher can move to $Room(u, \cdot, \cdot)$. □

The procedure in the proof of Lemma 9 is called *Local-Advance*. Let us return to the proof of the SECOND STEP. From the starting configuration $Room(c, p, q)$, we repeatedly apply Lemma 9 – to find a new cut $u \in \mathcal{N}$ and move to $Room(u, \cdot, \cdot)$ by the procedure *Local-Advance* – until either the left endpoint of $\mathcal{V}$ reaches q or the right endpoint of $\mathcal{V}$ reaches p. If the first event occurs before the second event, it means the 2-searcher has already cleared $Q(c)$, and so we can complete the SECOND STEP by simply moving the right endpoint of $\mathcal{V}$ clockwise to p. If the second event occurs before the first event, the right endpoint of $\mathcal{V}$ must be moving clockwise. When this execution of *Local-Advance* finishes, we have the configuration like this : the 2-searcher has cleared $D(r)$ for some $r \in \mathcal{N}$ and the *cw*-cave $Q(r)$ of r contains both p and q. See Figure 8. To complete the SECOND STEP, it suffices to show the following lemma.

Lemma 10 *Suppose that the current configuration is $Room(r, s', s)$ and that there is a cut $c \in \mathcal{N}$ such that c and r are contained in the cw-cave of each other. If N1 and N2 hold, the 2-searcher can clear $Q(c)$.*

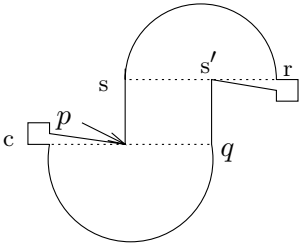


Fig. 8. Lemma 10.

Proof: See Figure 8. Suppose that $\mathcal{V}$ is currently at $\overline{s', s}$ and $D(r)$ is clear. First let us view $[p, s] \cup [s', q]$ as a corridor with doors $\overline{s's}$ and $\overline{qp}$. If it is 2-searchable we are done. Thus assume otherwise. If it contains "deadlock", we can find a u-triple. Thus some cut in the chains $[p, s]$ is not visible from the chain $[s', q]$, or vice versa.

To simplify the proof, we imagine that a pair of 2-searchers, say b and u, start to move from configurations $Room(c, p, q)$ and $Room(r, s', s)$, respectively, and show that they will meet at some time instant so that $\mathcal{V}_b$ and $\mathcal{V}_u$ correspond to each other. Then it follows that we can clear the corridor $[p, s] \cup [s', q]$ by first executing the movements of $\mathcal{V}_u$ and then executing the movements of $\mathcal{V}_b$ reversely.

Let L and R be subsets of $\mathcal{N}$, initially $L = \{c\}$ and $R = \{r\}$. If some cut in the chain $[p, s]$ is not visible from the chain $[s', q]$, take the most counterclockwise cut of $\mathcal{N}$ in $[p, s]$ whose *cw*-cave or *ccw*-cave contains $[s', q]$. (Symmetric arguments using Lemma 7 hold for the case that some cut in $[s', q]$ is not visible from $[p, s]$.) Let u denote it and assume that u is contained in $[p, s]$. First we claim that $[s', q]$ is contained in the *cw*-cave of u, because otherwise, that is, if it lies in the *ccw*-cave of u, then the three cuts c, r, and u form a u-triple. So q is in the *cw*-cave of u and by Lemma 9 $\mathcal{V}_b$ can advance to $Room(u, \cdot, \cdot)$. Moreover since the set $L \cup R \cup \{u\}$ must have a 2-cover (by N2), $D(u)$ must contain all cuts in L. We push u into L and update p and q to be the endpoints of $D(u)$.

We repeat the above procedure until the two boundary chains between $\mathcal{V}_b$ and $\mathcal{V}_u$ contain no cut that is not visible from the opposite chain. Why does not this procedure fall into an infinite loop? Notice that whenever u is picked in $[p, s]$, $D(u)$ must cover the cuts in L and especially cut c, to satisfy N2. This means the right endpoint of $\mathcal{V}_b$ cannot backtrack past $U(c)$ and thus the left endpoint of $\mathcal{V}_u$ cannot go past $U(c)$. Since one of $\mathcal{V}_b$ and $\mathcal{V}_u$ rotates each time but cannot go past some fixed point, this procedure will finish. □

4.4 First Step

We will show that if N1 and N3 hold, the 2-searcher can clear $D(c)$ for some cut $c \in \mathcal{N}$. By Lemma 7, it suffices to show that the 2-searcher can clear P, assuming that $Q(c)$ is clear. Actually this step is very similar to the SECOND STEP, except that we use N3 instead of N2.

Let us briefly describe the outline. Since the polygon satisfies N3, some essential cut has no s-pair; let g denote such a cut. Using the procedure *Local-Advance*, either the 2-searcher can clear $D(c)$ or she moves to the configuration $Room(x, p, q)$ such that g is contained in $(p, U(u)]$ for the most counterclockwise cut u of $\mathcal{N}$ in $Q(x)$ whose cw-cave contains q. We view $Q(x)$ as a corridor with two doors $\overline{pq}$ and $\overline{B(g)\,U(g)}$. If it is 2-searchable we are done. Assume otherwise. From the characterization of the class of searchable corridors, there are four cases and we can show that $Q(x)$ is searchable for each case. Details can be found in the full paper.

5 Concluding Remarks

There are some open problems. First our proof is quite complicated. One might want a simple proof (without using chracterizations) to show that the searcher with 360° vision and the 2-searcher have the same search capability. Some partial results can be found in [12,5]. Second, no results are known about computing a shortest search path in a searchable polygon, except [3].

Acknowledgement. We are grateful to anonymous referees for helpful comments.

References

1. D. Crass, I. Suzuki, and M. Yamashita. Searching for a mobile intruder in a corridor–the open edge variant of the polygon search problem. *Int. J. of Comp. Geom. and Appl.*, 5(4):397–412, 1995.
2. L.J. Guibas, J.C. Latombe, S.M. Lavalle, D. Lin, and R. Motwani. A visibility-based pursuit-evasion problem. *Int. J. of Comp. Geom. and Appl.*, 9(4):471–493, 1999.
3. C. Icking and R. Klein. The two guards problem. In *Proc. 7th Annu. ACM Sympos. Comp. Geom.*, pages 166–175, 1991.
4. S. M. LaValle, B. H. Simov, and G. Slutzki. An algorithm for searching a polygonal region with a flashlight. In *Proc. of 16th ACM Symp. on Comp. Geom.*, pages 260–269, 2000.
5. J.-H. Lee, S.-M. Park, and K.-Y. Chwa. On the Polygon-Search Conjecture. *Technical Report TR-2000-157, CS department, KAIST,* 2000.
6. J.-H. Lee, S.-M. Park, and K.-Y. Chwa. Searching a polygonal room with one door by a 1-searcher. *Int. J. of Comp. Geom. and Appl.*, 10(2):201–220, 2000.
7. J.-H. Lee, S.-M. Park, and K.-Y. Chwa. Simple algorithms for searching a polygon with flashlights. *Submitted*, 2000.
8. J.-H. Lee, S.Y. Shin, and K.-Y. Chwa. Visibility-based pursuit-evasion in a polygonal room with a door. In *Proc. 15th ACM Sympos. on Comp. Geom.*, pages 281–290, 1999.
9. N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *Journal of the ACM*, pages 18–44, 1988.
10. S.-M. Park, J.-H. Lee, and K.-Y. Chwa. Visibility-based pursuit-evasion in a polygonal region by a searcher. *Technical Report TR-2001-161, CS department, KAIST,* 2001.

11. T.D. Parsons. Pursuit-evasion in a graph. In *Theorey and Applications of Graphs Y. Alavi and D.R. Lick eds. Lecture Notes in Mathematics, Springer-Verlag.*, pages 426–441, 1976.
12. I. Suzuki, Y. Tazoe, M. Yamashita, and T. Kameda. Searching a polygonal region from the boundary. *TR-20000925, EECS Department, Univ. of Wisconsin-Milwaukee,* 2000.
13. I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM J. Comp.*, 21(5):863–888, 1992.
14. X. Tan. Searching a simple polygon by a k-searcher. In *Proc. of 11th ISAAC*, pages 503–514, 2000.
15. M. Yamashita, H. Umemoto, I. Suzuki, and T. Kameda. Searching for mobile intruders in a polygonal region by a group of mobile searchers. In *Proc. 13th Annu. ACM Sympos. Comp. Geom.*, pages 448–450, 1997.

A New Method for Balancing Binary Search Trees*

Salvador Roura

Departament de LSI, Universitat Politècnica de Catalunya,
E-08028 Barcelona, Catalonia, Spain.
roura@lsi.upc.es

Abstract. A new balancing method for binary search trees is presented, which achieves logarithmic worst-case cost on searches and updates. The method uses the sizes of the subtrees as balancing information; therefore operations by rank are efficiently performed without any changes in the data structure. Compared to weighted binary search trees [7], which also achieve logarithmic worst-case cost by making use of the sizes of the subtrees, the operations involved with our method are likely to be less costly in most real situations.

1 Introduction

The binary search tree (BST) data structure is fundamental to computer science. Since BSTs perform poorly when they are skewed, many variants of balanced BSTs have been devised so far. Weighted BSTs [7] achieve logarithmic worst-case cost by using the sizes of the subtrees as balancing information. Other variants, like AVL trees [1] and red-black trees [4], use information different from the sizes of the subtrees; thus rank operations are not efficiently supported unless an additional field is included at every node. The same comment applies to splay trees [9] and general balanced trees [2], which achieve logarithmic amortised costs without storing any structural information at the nodes. Other variants of balanced trees make use of the sizes of the subtrees but do not guarantee logarithmic worst-case cost; for instance, randomised BSTs [6].

This paper presents a new balancing method for BSTs, which, like weighted BSTs, achieves logarithmic worst-case cost by using the sizes of the subtrees as balancing information. So let us first briefly recall weighted BSTs. Suppose that L and R are the subtrees of a weighted BST, with x and y leaves respectively, and assume w.l.o.g. that $x \leq y$. The balancing property of weighted BSTs states that $y < (1+\sqrt{2})x$, or alternatively, that $2y^2 < (x+y)^2$, which is anyway an expensive property to check. This seems to be the main reason not to use weighted BSTs as default balancing method: "However, it appears that the bookkeeping required for maintaining weight balance takes more time than Algorithm A[1] ..." [5,

* This research was partially supported by the IST Programme of the EU IST-1999-14186 (ALCOM-FT), and by the project DGES PB98-0926 (AEDRI).

[1] Insertion in AVL trees.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 469–480, 2001.

page 476]. As shown in the next sections, the method introduced in this paper is likely to be more efficient than weighted BSTs in most practical situations.

The next sections are organised as follows. Section 2 introduces the main definitions, including the concept of Logarithmic BST (LBST), and proves that the height of an LBST is always logarithmical w.r.t. the size of the tree. Section 3 presents the insertion and deletion algorithms for LBSTs. The former is implemented in Sect. 4, where some empirical evidence that LBSTs are faster than weighted BSTS is provided. Section 5 ends the paper with some further comments.

2 Basic Definitions

Definition 1. *For positive n, let $\ell(n)$ be defined as*

$$\ell(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1 + \lfloor \log_2 n \rfloor, & \text{if } n \geq 1 \end{cases} .$$

Note that, except for $n = 0$, $\ell(n)$ is the largest position with a bit equal to one in the binary representation of n; in other words, $\ell(n)$ is the unique integer such that $2^{\ell(n)-1} \leq n \leq 2^{\ell(n)} - 1$.

Given a BST T, let $|T|$ denote the number of keys in T, and let $\ell(T) = \ell(|T|)$. We call our trees Logarithmic BSTs, since their fundamental property is that, at every node, the discrete logarithm of the size of the left subtree and the discrete logarithm of the size of the right subtree differ at most in one unit.

Definition 2 (Logarithmic BST). *A BST T is an LBST if and only if*

- *T is an empty tree,*
- *or T is a non-empty tree with subtrees L and R, such that L and R are LBSTs and $-1 \leq \ell(L) - \ell(R) \leq 1$.*

Let us consider the case where T is non-empty. Let $\lambda = \ell(L)$, and assume w.l.o.g. that $|L| \leq |R|$. Then $\ell(R)$ is either λ or $\lambda + 1$. We analyse both cases separately. Suppose first that $\ell(R) = \lambda$:

- If $\lambda > 0$, from $2^{\lambda-1} \leq |L|, |R| \leq 2^{\lambda} - 1$ we deduce that $2^{\lambda} + 1 \leq |T| \leq 2^{\lambda+1} - 1$. Therefore $\ell(T) = \lambda + 1$.
- If $\lambda = 0$, we have $|T| = 1$ and $\ell(T) = \lambda + 1$ as well.

Suppose now that $\ell(R) = \lambda + 1$:

- If $\lambda > 0$, the fact that $2^{\lambda-1} \leq |L| \leq 2^{\lambda} - 1$ and $2^{\lambda} \leq |R| \leq 2^{\lambda+1} - 1$ implies $3 \cdot 2^{\lambda-1} + 1 \leq |T| \leq 3 \cdot 2^{\lambda} - 1$. In this case $\ell(T)$ equals either $\lambda + 1$ or $\lambda + 2$.
- If $\lambda = 0$, then $|T| = 2$ and $\ell(T) = \lambda + 2$.

These two cases are summarised in Fig. 1, using the symbolism that we will keep for the rest of the paper. Every node is labeled with the name of the subtree rooted at that node. For every combination of the weights and every subtree S, the several possibilities for $\ell(S)$ are shown, unless $\ell(S)$ is the same in all situations (in that case it is shown just once). For example, in Fig. 1 we have that $\ell(L) = \lambda$ and $\ell(R) = \lambda$ imply $\ell(T) = \lambda + 1$, whilst $\ell(L) = \lambda$ and $\ell(R) = \lambda + 1$ imply $\ell(T) = \lambda + 1$ or $\ell(T) = \lambda + 2$.

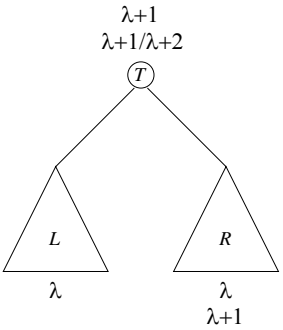


Fig. 1. Cases for an LBST with right subtree larger than left subtree

It is not difficult to prove that the height of an LBST T is always $\Theta(\log |T|)$. Let $\lambda = \ell(T)$. It is enough to notice that every grandchild G of T must satisfy $\ell(G) \leq \lambda - 1$. Otherwise we would have the situation of Fig. 2. Since both the brother of P and the brother of G would include at least $2^{\lambda-2}$ nodes, and G would include at least $2^{\lambda-1}$ nodes, T would have at least $2^{\lambda} + 2$ nodes, which is a contradiction. Therefore, every path from the root of T to a leaf visits at most $2(\lambda + 1)$ nodes, where $\lambda = \log_2 |T| + \mathcal{O}(1)$.

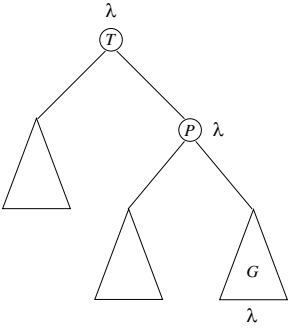


Fig. 2. Impossible case for an LBST

Theorem 3 below states that the constant 2 is in fact asymptotically tight, i.e., that the worst-case height of an LBST T is $\sim 2\log_2 |T|$. But first we need to introduce two functions. For every $N \geq 0$, define $I(N) = 4 \cdot 2^N + 2N$, and $J(N) = 6 \cdot 2^N + 2N + 1$. Observe that $4 = I(0) < J(0) < I(1) < J(1) < \cdots$ Hence, for every $n \geq 4$, there is a unique N such that either $I(N) \leq n < J(N)$ or $J(N) \leq n < I(N+1)$.

Theorem 3. *Let $H(n)$ be the maximum height of an LBST with n keys. Then $H(0) = 0$, $H(1) = 1$, $H(2) = H(3) = 2$, and for every $n \geq 4$,*

$$H(n) = \begin{cases} 2N+3, & \text{if } I(N) \leq n < J(N) \\ 2N+4, & \text{if } J(N) \leq n < I(N+1) \end{cases} .$$

(The proof is by induction on N.)

We thus know that, if $I(N) \leq n < I(N+1)$, $H(n) \leq 2(N+2)$. But $N + 2 \leq \log_2 I(N) \leq \log_2 n$, and we can conclude that the height of an LBST with $n \geq 2$ keys is never larger than $2\log_2 n$. Note that the constant 2 is asymptotically tight, i.e., $H(n) \sim 2\log_2 n$. Recall that the worst-case height of a weighted BST with n keys is also $\sim 2\log_2 n$.

3 The Insertion and Deletion Algorithms

In the insertion and deletion algorithms, we will make use of the following algorithm, which obtains an LBST from a BST such that its left subtree L and its right subtree R are LBSTs, where $\ell(L) = \lambda - 2$ and $\ell(R) = \lambda$. Let A and D be the left and right subtrees of R, respectively. Figure 3 includes the five possible combinations for $\ell(A)$ and $\ell(D)$, provided that $\ell(R) = \lambda$.

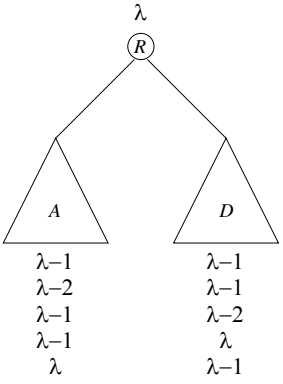


Fig. 3. Five possible cases for an LBST R with $\ell(R) = \lambda$

As shown in Fig. 4, a single rotation suffices for the first, second and fourth cases of Fig. 3. For instance, consider the first case, where $\ell(A) = \ell(D) = \lambda - 1$.

After the rotation, the left subtree of R, labeled T in the figure, is such that $\lambda - 1 \leq \ell(T) \leq \lambda$, and thus $\ell(T)$ differs in at most one unit with $\ell(D)$.

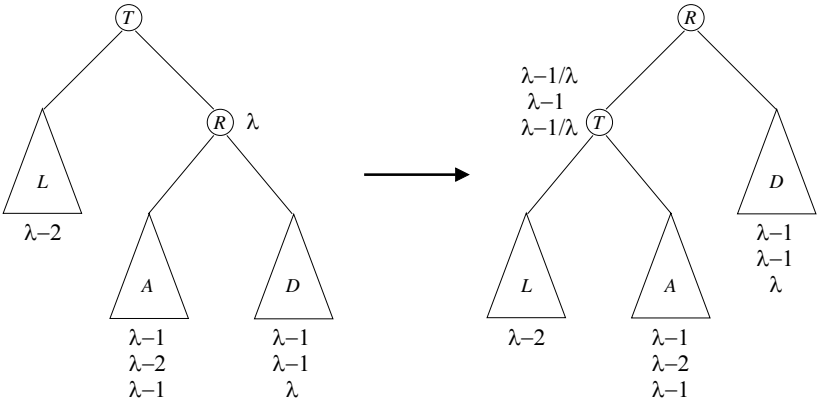


Fig. 4. Cases of Fig. 3 for which a single rotation suffices

Figure 5 proves that, for the third case of Fig. 3, a double rotation suffices. Let B and C be the left and right subtrees of A, respectively (notice that $\ell(A) \geq 1$ implies that A is never empty). As in Fig. 3, there are five possible combinations for $\ell(B)$ and $\ell(C)$. For each one and after two rotations, the first rotation between A and R, the second between A and T, the balancing property is reestablished.

The fifth and last case of Fig. 3 also requires a double rotation (see Fig. 6), but this case is slightly different from the case in Fig. 5. Indeed, only three of the five combinations for $\ell(B)$ and $\ell(C)$ are possible here, since $\ell(B) = \lambda - 1$ and $\ell(C) = \lambda$ (or $\ell(B) = \lambda$ and $\ell(C) = \lambda - 1$) together with $\ell(D) = \lambda - 1$ would imply $\ell(R) = \lambda + 1$, which is against the hypotheses.

We are now ready to present the insertion algorithm of a new key x into a given LBST T, which follows the traditional approach of balanced trees:

- If T is empty, return a BST with x as only key.
- Otherwise, let L and R be the left and right subtrees of T, respectively.
  - If x is smaller than the root of T, recursively insert x into L; if afterwards $\ell(L) = \ell(R) + 2$, perform a local update.
  - If x is larger than the root of T, recursively insert x into R; if afterwards $\ell(R) = \ell(L) + 2$, perform a local update.

The local updates mentioned above, meant to reestablish the balancing property of LBSTs, are those included in Figs. 4, 5 and 6. Note that, in fact, only the second and third cases of Fig. 3 are possible here, because $|R|$ must be exactly $2^{\lambda-1}$ after the recursive insertion.

We now consider how to delete a key x from a given LBST T. The deletion algorithm also uses the local updates included in Figs. 4, 5 and 6:

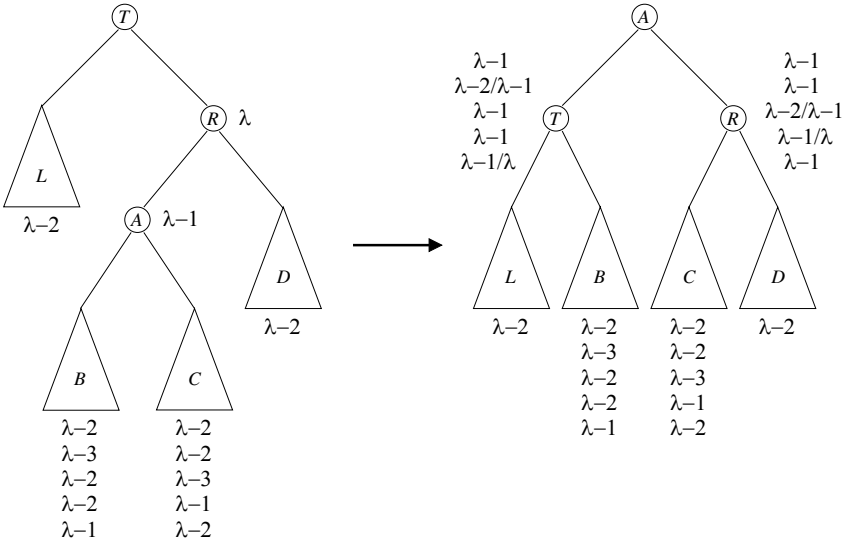


Fig. 5. Third case of Fig. 3; a double rotation suffices

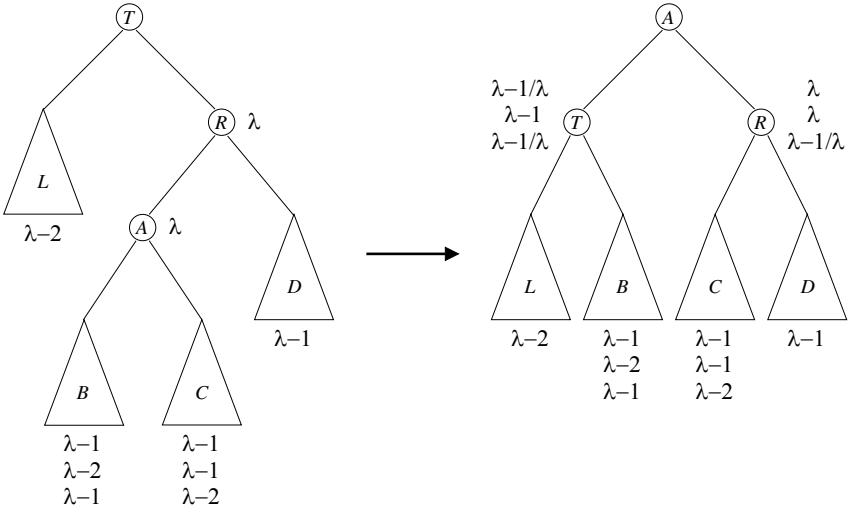


Fig. 6. Fifth case of Fig. 3; a double rotation suffices

- If T is empty, x is not in T; hence no updates are needed.
- If T has x as unique key, return the empty tree.
- Otherwise, let L and R be the left and right subtrees of T, respectively.
  - If x is smaller than the root of T, recursively delete x from L; if afterwards $\ell(R) = \ell(L) + 2$, perform a local update.
  - If x is larger than the root of T, recursively delete x from R; if afterwards $\ell(L) = \ell(R) + 2$, perform a local update.
  - If x is equal to the root of T, remove it.

The removal of the root of T can be done in several ways. For instance, we can replace the root of T by the minimum of the keys in R when $|L| \leq |R|$, or by the maximum of the keys in L when $|L| > |R|$. The algorithm to extract the minimum of the keys from a non-empty LBST T is quite simple:

- If the left subtree of T is empty, the minimum key is the root of T, and the right subtree of T is the remaining tree.
- Otherwise, recursively extract the minimum from the left subtree of T; afterwards perform a local update if necessary.

Once again, the local updates required here are identical to those of the insertion algorithm and deletion algorithm. The algorithm to extract the maximum of the keys of a non-empty LBST is symmetrical.

4 Implementing LBSTs

The C code presented in this paper implements the insertion algorithm for LBSTs. This code has been written to emphasise the simplicity of the algorithms, so faster programmes could be obtained at the price of obscuring the code. Due to space limitations the deletion algorithm has been omited.

As shown in Fig. 7, an LBST is identified with a pointer to its root node. Every node contains a key, two pointers to its children and a counter of type `size` with the number of keys in the subtree rooted at the node. We assume that keys and counters are long integers, and that empty trees are equal to `null`, which is a pointer to a node with no key and size 0. The call `singleton(x)` returns an LBST with `x` as only key.

Figure 7 also includes some fundamental functions. Given two sizes `a` and `b`, the call `smaller_ell(a, b)` tells us whether $\ell(\mathtt{a}) < \ell(\mathtt{b})$ or not. Let $\lambda = \ell(\mathtt{b})$. If $\mathtt{a} \geq \mathtt{b}$, we trivially have $\ell(\mathtt{a}) \geq \lambda$. Otherwise, we perform a logical "and" of `a` and `b`, shift the result one bit to the left, and compare the final result (let us call it α) against `b`. Assume that $\lambda \geq 1$. If $\ell(\mathtt{a}) = \lambda$, we have $\ell(\alpha) = \lambda + 1$; hence $\alpha > \mathtt{b}$ and the function returns `FALSE`. If $\ell(\mathtt{a}) < \lambda$, then α is at most $2(\mathtt{b} - 2^{\lambda-1})$. This happens when $\ell(\mathtt{a}) = 2^{\lambda-1} - 1$, i.e., when the digital representation of `a` includes as many bits equal to one as possible. Since $\mathtt{b} < 2^{\lambda}$, we have $\alpha \leq 2\mathtt{b} - 2^{\lambda} < \mathtt{b}$, and the function returns `TRUE`, es expected. The function always returns `FALSE` for the special case $\lambda = 0$.

The call `rot_left(t)` returns the result of rotating `t` to its left, updating conveniently the fields `b->s` and `t->s`. The call `inc_left(t)` returns the result

of balancing `t` by means of the rotations in Figs. 4, 5 and 6, assuming that ℓ(`t->r`) = ℓ(`t->l`) + 2. Notice that one call to the function `smaller_ell()` suffices to discriminate the first, second and fourth cases from the third and fifth cases of Fig. 3. The functions `rot_right()` and `inc_right()` are easily obtained from the functions `rot_left()` and `inc_left()`.

Given two sizes `A` and `B` such that ℓ(`A`) $\leq$ ℓ(`B`) + 1, we use the macro `balanced(A, B)` to know whether ℓ(`B`) $\leq$ ℓ(`A`) + 1 or not. Note that the logical instructions in this macro and in the function `smaller_ell()` are usually fast in most computers.

```
typedef long key, size;
typedef struct node *lbst;
typedef struct { key k; lbst l, r; size s; } node;

int smaller_ell(size a, size b)
{ if (a >= b) return FALSE;
  return ((a&b)<<1) < b;
}

lbst rot_left(lbst t)
{ lbst b = t->r; t->r = b->l; b->l = t;
  b->s = t->s; t->s = 1 + t->l->s + t->r->s;
  return b;
}

lbst inc_left(lbst t)
{ if (smaller_ell(t->r->r->s,t->r->l->s)) t->r = rot_right(t->r);
  return rot_left(t);
}

#define balanced(A, B) !(smaller_ell(A, (B)>>1))

lbst Insert(key x, lbst t)
{ if (t == null) return singleton(x);
  t->s++;
  if (x < t->k)
    { t->l = Insert(x, t->l);
      if (!balanced(t->r->s, t->l->s)) t = inc_right(t);
    }
   else
    { t->r = Insert(x, t->r);
      if (!balanced(t->l->s, t->r->s)) t = inc_left(t);
    }
  return t;
}
```

Fig. 7. Insertion algorithm in *C*

The left side of Table 1 shows the empirical average search cost and height of an LBST produced after n random insertions into an initially empty tree, for several values of n. Table 1 also includes the total number of single and double rotations that take place during the construction of the LBST. The first four rows are averaged over 100 executions; the last four rows are averaged over 10 executions. The right side of Table 1 includes the same measures, this time for LBSTs built in increasing order. We define the average search cost as the internal path length divided by the number of keys.

Table 1. Empirical average search cost, height, and number of single and double rotations of LBSTs built under random and sorted insertions

	Random order				Increasing order			
# keys	A.S.C.	Height	# S.R.	# D.R.	A.S.C.	H.	# S.R.	# D.R.
15625	13.254	17.60	3415.78	3413.68	12.952	15	15611	0
31250	14.276	18.97	6829.06	6822.12	13.952	16	31235	0
62500	15.290	20.15	13636.65	13660.82	14.952	17	62484	0
125000	16.316	21.45	27336.78	27282.39	15.952	18	124983	0
250000	17.340	22.8	54678.0	54634.3	16.952	19	249982	0
500000	18.366	23.9	109496.4	109073.1	17.951	20	499981	0
1000000	19.376	25.3	218454.2	218615.9	18.951	21	999980	0
2000000	20.379	26.3	436917.6	437033.4	19.951	22	1999979	0

It is not difficult to prove that any LBST obtained after inserting the keys in increasing order is (almost) perfectly balanced (and hence the results for sorted insertions given in Table 1). An exact analysis of random LBSTs is much harder. However, the empirical results provided in Table 1 indicate that the average search cost is $\sim \beta \cdot \log_2 n$ for some constant β very close to 1, which can be regarded as optimal for practical purposes. A similar result holds for other variants of balanced BSTs.

A single insertion into an LBST T may require up to $\Theta(\log |T|)$ rotations. However, less than n rotations are enough to build in increasing order an LBST with n keys, and, from Table 1, the total number of rotations under random insertions also seems to be $\Theta(n)$. The next theorem states that this is not a coincidence.

Theorem 4. *The total number of rotations required to build an LBST T from an empty tree is $\mathcal{O}(|T|)$.*

(The theorem can be proved by means of the potential method; the author's proof is too long to be included in this paper. Note that the same property is true for weighted BSTs [3].)

Tables 2 and 3 include some empirical results about the time efficiency of our algorithms. The tests consisted in the construction of BSTs with n keys, for five different balancing strategies and several values of n. Two limiting situations were considered, namely when keys are inserted at random (Table 2), and when

keys are inserted in increasing order (Table 3). The times, expressed in seconds, were obtained with a PC[2], and averaged over 1000 executions for the first four rows, and over 100 executions for the last four rows.

Table 2. Empirical times (in seconds) to build LBSTs, proper weighted BSTs, relaxed weighted BSTs, AVL trees and red-black tres in random order

# keys	LBSTs	WBSTs	3WBSTs	AVLs	RBTs
15625	0.01424	0.01711	0.01437	0.01791	0.02811
31250	0.04909	0.05457	0.04933	0.05543	0.07658
62500	0.14171	0.15370	0.14291	0.15323	0.19947
125000	0.36025	-	0.36261	0.40145	0.48025
250000	0.8667	-	0.8694	0.9547	1.1331
500000	2.0340	-	2.0344	2.2339	2.5683
1000000	4.6638	-	4.6783	5.1646	5.7938
2000000	10.6002	-	10.6326	11.6718	13.0526

The code used for LBSTs was the one provided in this paper, with the function `smaller_ell()` replaced by a macro for efficiency. The code used for weighted BSTs was the same except for the balancing condition, which was "$2y^2 < (x+y)^2$", where x and y are respectively the number of leaves of the "small" subtree and of the "large" subtree (this condition was only checked after an insertion into the "large" subtree). The computation of $2y^2$ and $(x+y)^2$ caused an overflow for large values of n; hence the empty fields in Tables 2 and 3. A relaxed variant of weighted BSTs was also implemented, with "$y < 3x$" as balancing condition (we call these trees 3WBSTs). The property for 3WBSTs is cheaper to check than the one for proper weighted BSTS, but it can degrade somehow the tree, since the worst-case height becomes $\sim \ln 2/\ln(4/3)\cdot\log_2 n \simeq 2.40942\log_2 n$. The code for AVL trees and the code for red-black trees were taken from [10, page 153] and from [8, page 554] respectively. Both codes were slightly modified, to make them comparable with the code of the rest of balancing strategies.

Under random insertions, LBSTs and 3WBSTs performed similarly, and faster than WBSTs. Since the trees obtained are very well balanced in all the cases, the crucial factor was the high cost of evaluating the balancing property of WBSTs. For sorted insertions, both WBSTs and 3WBSTs were about 30 percent slower than LBSTs; note that 3WBSTs built in increasing order are not perfectly balanced. In general, red-black trees achieved the worst times, while AVL trees turned out to be about 10 percent slower than LBSTs. The implementation of AVL trees was the only one without a counter field with the sizes of the subtrees. If rank operations were needed, this extra field should be updated conveniently during the insertions, which would increase the insertion time of AVL trees. Note that the time of sorted insertions was much smaller than the time of random insertions. This was probably due to the high memory locality of the former, which induced an efficient use of the cache memory.

[2] Pentium(r) II Processor, 128 MB of RAM, DJGPP *C* compiler.

Table 3. Empirical times (in seconds) to build LBSTs, proper weighted BSTs, relaxed weighted BSTs, AVL trees and red-black tres in increasing order

# keys	LBSTs	WBSTs	3WBSTs	AVLs	RBTs
15625	0.01125	0.01467	0.01474	0.01253	0.03040
31250	0.02651	0.03447	0.03431	0.03018	0.06751
62500	0.05939	0.07635	0.07672	0.06719	0.14485
125000	0.12968	0.16485	0.16570	0.14448	0.31201
250000	0.2746	-	0.3541	0.3072	0.6568
500000	0.5844	-	0.7600	0.6559	1.3985
1000000	1.2266	-	1.6241	1.3908	2.9786
2000000	2.5781	-	3.4708	2.9299	6.3003

5 Final Remarks

Other operations for BSTs, like joins, splits, unions, intersections, set subtractions, and so on, can be easily and efficiently implemented using the ideas in this paper. Moreover, since a counter field is kept at each node, rank operations are efficiently performed without any further modification of our data structure.

There are several variants of LBSTs that may be considered. First, it is possible to use the number of leaves instead of the number of keys as balancing information. The algorithms obtained perform similarly to the ones presented in this paper. On the other hand, we could relax the condition in Definition 2 to be $-k \leq \ell(L) - \ell(R) \leq k$ for some constant $k \geq 1$; alternatively, we could define $\ell(n) = 1 + \lfloor \log_b n \rfloor$ for some base $b \neq 2$, or combine several of these possibilities. In general, LBSTs with large k (or with large b) perform less rotations than plain LBSTs, since its balancing condition is less astringent. On the other hand, its worst-case height increases as k (or b) increases.

Finally, it must be said that the experimental results presented in this paper are only an indication that LBSTs can be a practical alternative to traditional balancing strategies. Nevertheless, there are many factors that should be considered in our election: time (and space) efficiency, algorithm and code complexity, variety of supported operations (rank operations, set operations, etc.), ease of obtaining non-recursive versions to increase efficiency, average and worst-case cost (measured as number of visited nodes, rotations, etc.), and so on.

Acknowledgments. The comments of Josep Díaz, Rolf Fagerberg and Conrado Martínez improved the presentation of this work.

References

[1] G.M. Adel'son-Vel'skii and E. M. Landis. An algorithm for the organization of information. *Dokladi Akademia Nauk SSSR*, 146(2):263–266, 1962. English translation in Soviet Math. Doklay 3, 1259-1263, 1962.

[2] A. Andersson. General balanced trees. *Journal of Algorithms*, 30:1–18, 1999.

[3] N. Blum and K. Mehlhorn. On the average number of rebalancing operations in weight-balanced trees. *TCS: Theoretical Computer Science*, 11:303–320, 1980.
[4] L.J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proc. of the 19th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 8–21, October 1978.
[5] D.E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, Reading, MA, 2nd edition, 1998.
[6] C. Martínez and S. Roura. Randomized binary search trees. *Journal of the ACM*, 45(2):288–323, March 1998.
[7] J. Nievergelt and E. Reingold. Binary search trees of bounded balance. *SIAM Journal on Computing*, 2(1):33–43, 1973.
[8] R. Sedgewick. *Algorithms in C*. Addison-Wesley, 3rd edition, 1998.
[9] D.D. Sleator and R.E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, July 1985.
[10] M.A. Weiss. *Data Structures & Algorithm Analysis in C++*. Addison-Wesley, 2nd edition, 1999.

Permutation Editing and Matching via Embeddings

Graham Cormode[1], S. Muthukrishnan[2], and Süleyman Cenk Ṣahinalp[3]

[1] University of Warwick, Coventry, UK; grahamc@dcs.warwick.ac.uk
[2] AT&T Research, Florham Park, NJ, USA; muthu@research.att.com
[3] EECS, Case Western Reserve University, Cleveland, OH; cenk@cwru.edu

Abstract. If the genetic maps of two species are modelled as permutations of (homologous) genes, the number of chromosomal rearrangements in the form of deletions, block moves, inversions etc. to transform one such permutation to another can be used as a measure of their evolutionary distance. Motivated by such scenarios, we study problems of computing distances between permutations as well as matching permutations in sequences, and finding most similar permutation from a collection ("nearest neighbor").

We adopt a general approach: embed permutation distances of relevance into well-known vector spaces in an approximately distance-preserving manner, and solve the resulting problems on the well-known spaces. Our results are as follows:

- We present the first known approximately distance preserving embeddings of these permutation distances into well-known spaces.
- Using these embeddings, we obtain several results, including the first known efficient solution for approximately solving nearest neighbor problems with permutations and the first known algorithms for finding permutation distances in the "data stream" model.
- We consider a novel class of problems called *permutation matching* problems which are similar to string matching problems, except that the pattern is a permutation (rather than a string) and present linear or near-linear time algorithms for approximately solving permutation matching problems; in contrast, the corresponding string problems take significantly longer.

1 Introduction

As the first phase of the Human Genome Project approaches completion, the attention is shifting from raw sequence data to genetic maps. Comparative studies of gene loci among closely related species provide clues towards understanding the complex phylogenetic relationships between species and their evolutionary order. Genetic maps of two species can be thought of as permutations of homologous genes and the number of chromosomal rearrangements in the form of deletions, copies, inversions, transpositions to transform one such permutation to another can be used as a measure of their evolutionary distance. Computational methods for measuring genetic distance between species is an active area

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 481–492, 2001.

of research in computational genomics, especially in the context of comparative mapping [13], eg., using reversal distance [1,4,11], transposition distance [2,8] or other measures. In a more general setting it is of interest to not only compute the distances between two permutations but also to find the closest gene permutation to a given one in a database or to approximately find a given permutation of genes in a larger sequence, etc. Given the representation as permutations, these can all be abstracted as permutation editing and matching problems.[1]

Permutations are ordered sequences over some alphabet with no repetitions allowed.[2] Thus, any permutation is a string, although strings are not generally permutations since they are allowed to repeat symbols. Suitable edit operations on permutations include reversals; transpositions; alphabet edits such as inserts and deletes; and symbol moves (formal definition of these operations follows). We study problems of computing pairwise edit distances, similarity searching, matching and so on, motivated by Computational Biology and other scenarios. We adopt a general approach to solving all such problems on permutations: develop an embedding of permutations into vector spaces such that the distance between the resulting vectors approximates the distance between any two permutations. Thus permutation editing and matching problems reduce to natural problems on vector spaces.

Even though we have motivated permutation editing, matching and similarity searching problems from Computational Biology applications, there are other reasons for their study. Permutations form an interesting class of combinatorial objects by themselves, and therefore it is quite natural to study the complexity of computing edit distances between permutations, and to do similarity searching. In addition, they arise in many applications. Since permutations are special cases of strings, permutation editing and matching problems give insight into the complexity of string editing and matching problems many of which are classical and still open. This will be clarified later using our results. In what follows, we will first describe the edit distance problems with permutations before describing our results.

1.1 Notation

A permutation is a sequence of symbols such that within a permutation each symbol is unique. We shall often represent these symbols as integers drawn from some range, so 1 3 2 4 is a valid permutation, but 1 2 3 2 is not. Signed permutations are permutations where each symbol can take two forms: positive and negative, eg $1^+\ 3^+\ 2^-\ 4^+$. Operations can also change the signs of symbols, and two signed permutations are considered identical only if every symbol and every sign agree. In what follows, P, Q will represent permutations, and $i, j, k \ldots$ will

[1] More complex notions of genetic distance which take into account that (1) the genome is composed of multiple chromosomes [6,14], or (2) exact order of the genes within a genome is not necessarily known [7] have recently been proposed.

[2] Sometimes it matters in which orientation a gene occurs, and so use is made of *signed permutations*

be integers. The i'th symbol of a permutation P will be denoted as $P[i]$, and the inverse of the permutation P^{-1} is defined so that if $P[i] = j$ then $P^{-1}[j] = i$. We can also compose one permutation with another, so $(P \circ Q)[i] = P[Q[i]]$. The "identity permutation" is the permutation of for which $P[i] = i$ for all i. For uniformity, we shall extend *all* permutations P by adding $P[0] = 0$ and $P[n+1] = n+1$, where n is the length of P. This allows the first and last symbols of P to be treated identically to the other symbols. All logarithms will be taken to base 2, and rounded up, so $\log n$ should be interpreted as $\lceil \log_2 n \rceil$.

1.2 Permutation Editing and Matching Problems

First we focus on defining distances between permutations. Consider any two permutations P and Q over some alphabet set. The following distances are of interest:

Reversal Distance: Denoted $r(P, Q)$, reversal distance is defined as the minimum number of reversals of contiguous subsequences necessary to transform permutation P into Q. So if P is a permutation $P[1] \ldots P[n]$, then a Reversal operation with parameters i, j $(i < j)$ results in the permutation $P[1] \ldots P[i-1], P[j], P[j-1] \ldots P[i+1], P[i], P[j+1] \ldots P[n]$. If P is a signed permutation then additionally the sign of each symbol $P[j] \ldots P[i]$ is switched (from plus to minus and vice-versa). This distance has been well-studied, and is shown to be NP-hard to find exactly [3]. The best approximation algorithm for this distance is a 3/2 factor algorithm due to Christie [4].

Transposition Distance: Denoted $t(P, Q)$, transposition distance is defined as the minimum number of moves of contiguous subsequences to arbitrary new locations necessary to transform permutation P into Q. Bafna and Pevzner [2] give a 3/2 approximation algorithm for transposition distance. Given $P[1] \ldots P[n]$, a transposition with parameters i, j, k $(i < j < k)$ gives
$P[1] \ldots P[i-1], P[j], P[j+1] \ldots P[k], P[i], P[i+1] \ldots P[j-1], P[k+1] \ldots P[n]$.

Permutation Edit Distance: The permutation edit distance between two permutations, $d(P, Q)$ is the minimum number of moves required to transform P into Q. A move can take a single symbol and place it at an arbitrary new position in the permutation. Hence a move with parameters i, j $(i < j)$ turns $P[1] \ldots P[n]$ into $P[1] \ldots P[i-1], P[i+1] \ldots P[j], P[i], P[j+1] \ldots P[n]$. This distance is analogous to the Levenshtein edit distance on strings, since in both cases an optimal set of edit operations will isolate a longest common subsequence and leave this unaltered, while performing edit operations on every other symbol.

Symbol Indels: Each of the above distances can be augmented by additionally allowing insertions and deletions of a single symbol at a time. This takes care of the fact that the alphabet set in two permutations need not be identical.

It will be of interest to (1) combine all operations (transposition, reversal, symbol moves) and define the cumulative distance between any two permutations involving minimum number of operations, and (2) generalize the definitions so that at most one of P or Q is a string (as opposed to a permutation). If both P and Q are strings, we are in the familiar territory of string matching.

1.3 Our Results

Our main results are threefold. We give them in outline; the precise bounds are given in later sections.

In Section 2 we present embeddings of permutation distances into well-understood spaces such as Hamming or Set Intersection. The embeddings preserve the original distances within a small constant or logarithmic factor, and are small polynomial in size. These are the first such approximately distance-preserving embeddings in the literature for permutation distances. The embeddings use a technique we develop in this paper of capturing the relative layout of pairs of symbols in the permutations by two dimensional matrices. Our embeddings capture the relevant pairs that help approximate the permutation distances accurately and the resulting matrices are often sparse. We believe that this approach to embedding distances will be of independent interest.

The embeddings above immediately give approximation algorithms for computing the distance between two permutations in (near) linear time. In addition, we use the embeddings above to solve several other algorithmic problems of which we list the following two as important examples: (1) Computing permutation distances in a distributed or Communication Complexity setting wherein the number of bits exchanged is the prime criterion for efficiency, and also in a "data streaming" model wherein data is scanned in order as it streams by and only small amount of space is allotted for storage. Streaming algorithms are known for vector distance computations; nothing was known beforehand for permutations — moreover, no streaming algorithms were known for any string distances. (2) Providing efficient approximate nearest neighbor searches for permutation distances. We provide the first known approximate algorithms for these problems that avoid the dimensionality bottleneck. These are all described in Section 3.

The problem of *Approximate Permutation Matching* is, given a long text string and a pattern permutation, to find all occurrences of the pattern in the text with at most a given threshold of distance. This is the generalization of the standard k-mismatches problem with strings (find all text locations wherein the pattern occurs with at most k mismatches) to other edit distances, and a restriction since the pattern is required to be a permutation. In Section 4 we present highly efficient, linear or near-linear time approximations for the permutation matching problems. This is intriguing since approximately solving string matching problems with corresponding distances seems to be harder, with best known algorithms taking much longer. For example, approximating string matching with edits takes time $\Omega(nm)$ for n-long text and m-long pattern where edits are transpositions, character indels and substitutions, and at least $\Omega(n \log^3 m)$ even if only substitutions are allowed [10]! In contrast, our algorithms take only $O(n + m)$ or $O(n \log m)$ time for permutations.

Our embeddings give other results such as efficient clustering algorithms for permutations, and other similarity problems. We do not discuss them further here since they follow in a straightforward way by combining our embeddings with results known for the target spaces such as Hamming, L_1, and Set Intersection. This is a welcome side-effect of our use of embeddings.

2 Embeddings of Permutation Distances

2.1 Reversal Distance

For signed permutations, we replace every positive element i^+ with the pair $i'\ i''$ and every i^- with $i''\ i'$. The reversal distance of two unsigned versions is the same as the reversal distance of the original permutations. We define a two dimensional matrix, $R(P)$, as a binary matrix of size $(n+2) \times (n+2)$. For all $0 \leq j < i \leq n+1$, set $R(P)[i,j]$ to 1 if $i > j$ and i is adjacent to j in P, that is, if either $P^{-1}[i] = 1 + P^{-1}[j]$ or $P^{-1}[j] = 1 + P^{-1}[i]$. Otherwise, $R[i,j] = 0$. We set $R[i,i] = 0$ for all i, and the matrix is only populated above this main diagonal. Recall that the reversal distance between two permutations is denoted as $r(P,Q)$. The Hamming distance between two bit vectors X and Y is denoted $H(X,Y)$. The Hamming distance between two matrices is the Hamming distance between two vectors obtained by linearizing the two matrices in any manner.

Theorem 1. $r(P,Q) \leq \frac{1}{2}H(R(P),R(Q)) \leq 2r(P,Q)$

Proof. We extend the notion of Reversal Breakpoints given in Section 2 of [11] which is defined on a single permutation. Define a *Reversal Breakpoint of P relative to Q* as a location, i, where the symbol following $P[i]$ in P is not adjacent to $P[i]$ where it occurs in Q. Formally, this is when $|Q^{-1}[P[i]]-Q^{-1}[P[i+1]]| \neq 1$. We denote the total number of such breakpoints as $\phi(P,Q)$. Clearly, if $P = Q$, then $\phi(P,Q) = 0$, and this is the only way in which the count is zero. In transforming P into Q using reversals, our goal is to reduce ϕ to zero. A reversal affects two locations, so we can reduce ϕ by at most two per move, which gives a lower bound. It is also the case that we can always convert P into Q using at most $\phi(P,Q)$ reversals. This follows from considering relabelling Q as the identity permutation, and applying this same relabelling to P generating $Q^{-1} \circ P$. The reversal breakpoints of P relative to Q then become precisely the reversal breakpoints of $Q^{-1} \circ P$ relative to the identity permutation, and consequently, the permutation can be edited using at most this number of reversals, following from Theorem 1 in [11]. Hence $r(P,Q) \leq \phi(P,Q) \leq 2r(P,Q)$.

It remains to show that $H(R(P),R(Q)) = 2\phi(P,Q)$. Suppose that $R(P)[i,j] = 1$ and $R(Q)[i,j] = 0$. This means that i and j are adjacent in P but not in Q. If we sum the number of distinct pairs i,j which are adjacent in P but not in Q, then this finds $\phi(P,Q)$. This is because every breakpoint will generate such a pair, and such pairs can only arise from breakpoints. An identical argument follows when $R(P)[i,j] = 0$ and $R(Q)[i,j] = 1$, yielding $\phi(Q,P)$. Since $\phi(Q,P) = \phi(P,Q)$, it follows that $H(R(P),R(Q))$ counts each breakpoint exactly two times.

2.2 Transposition Distance

We define $T(P)$, a binary matrix for a permutation P such that $T(P)[i,j] = 1$ if j immediately follows i in P, ie if $P^{-1}[i] + 1 = P^{-1}[j]$.

Theorem 2. $t(P,Q) \leq \frac{1}{2}H(T(P),T(Q)) \leq 3t(P,Q)$

Proof. Define a *Transposition Breakpoint* in a permutation P relative to another permutation Q as a location, i, such that $P[i+1]$ does not immediately follow $P[i]$ when it occurs Q,[3] that is $Q^{-1}[P[i]]+1 \neq Q^{-1}[P[i+1]]$. Let the total number of such transposition breakpoints between P and Q be denoted as $tb(P,Q)$. Observe that to convert P to Q we must remove all breakpoints, since $tb(Q,Q) = 0$. A single transposition affects three locations and so could 'fix' at most three breakpoints — this gives the lower bound. Also, we can always fix at least one breakpoint per transposition using the trivial greedy algorithm, which gives the upper bound. Hence $t(P,Q) \leq tb(P,Q) \leq 3t(P,Q)$.

We now need to show that $H(T(P),T(Q)) = 2tb(P,Q)$: clearly, $tb(P,Q) = tb(Q,P)$. $T(P)[i,j] = 1$ and $T(Q)[i,j] = 0$ if and only if there is a transposition breakpoint in Q at the location of i, so summing these contributions generates $tb(P,Q)$. A symmetrical argument holds when $T(P)[i,j] = 0$ and $T(Q)[i,j] = 1$, and these two cases summed generate exactly $H(T(P),T(Q)) = 2tb(P,Q)$.

2.3 Permutation Edit Distance

We show how to embed Permutation Edit Distance into Set Intersection Size up to a factor of $\log n$. We shall define $A(P)$ as an $n \times n$ binary matrix derived from a permutation of length n, P. $A_k(P)[i,j]$ is set to one if symbol i occurs a distance of exactly 2^k before j in P. Otherwise, $A_k(P)[i,j] = 0$. $A(P)$ is formed by taking the union of the matrices $A_0 \ldots A_{\log n-1}$. That is, $A(P)[i,j] = 1 \iff \exists k.\ (P^{-1}[i]+2^k = P^{-1}[j])$. Note that $A(P)$ is a binary matrix, and $n \log n + \Theta(n)$ entries are 1.

Also, let $B(Q)$ be an $n \times n$ binary matrix defined on a permutation Q such that $B(Q)[i,j]$ is zero if i occurs before j in Q. Otherwise $B(Q)[i,j] = 1$. Thus, $B(Q)[i,j] = 0 \iff (Q^{-1}[i] < Q^{-1}[j])$ In this matrix, $n^2/2 + \Theta(n)$ entries are 1. Finally, define $D(P,Q)$ as the size of the intersection between $A(P)$ and $B(Q)$. Put another way, this intersection can be calculated using multiplication of the elements of the matrices, pairwise: $D(P,Q) = \sum_{i,j}(A(P)[i,j] \times B(Q)[i,j])$.

Theorem 3. $d(P,Q) \leq D(P,Q) \leq \log n \cdot d(P,Q)$.

Proof. i) $D(P,Q) \leq \log n \cdot d(P,Q)$
Consider the pairs (i,j) such that $A(P)[i,j] = B(Q)[i,j] = 1$. The number of such pairs is exactly $D(P,Q)$. Each of these pairs has i occurring before j in P, but the other way round in Q, and so one of either i or j must be moved to turn P into Q. So in effect, these pairs represent a "to-do" list of changes that must be made. By construction of A, any symbol i appears at most $\log n$ times amongst these pairs. Hence whenever a move is made, at most $\log n$ pairs can be removed from this to-do list. It therefore follows that in each move, D can change by at most $\log n$. If at every step we change D by at most $\log n$, then this bounds the minimum number of operations possible to transform P into Q as $D(P,Q)/\log n \leq d(P,Q)$

[3] As usual, we extend all permutations so that the first symbol is 0 and their last is $n+1$.

ii) $d(P,Q) \leq D(P,Q)$
We shall show the bound by concentrating on the fact that an optimal edit sequence preserves a Longest Common Subsequence of the two sequences. Note that an optimal edit sequence will have length $n - LCS(P,Q)$: every symbol that is not moved must form part of a common subsequence of P and Q and so an optimal edit scheme will ensure that this common subsequence is as long as possible. Consider the relabelling of Q so that for all i, $Q[i]$ is relabelled with i. We analyze the effect of applying this relabelling to P and examine its longest increasing subsequence. Call this relabelled sequence P'. Clearly, the longest common subsequence of P and Q is not altered, since we have just relabelled distinct symbols. Because Q is replaced by a strictly increasing sequence, it follows that each Longest Common Subsequence of P and Q corresponds exactly to one Longest Increasing Subsequence of P', whose length is denoted by $LIS(P')$. Qualitatively, what D told us was that we count 1 if symbol is 2^k to the right of the i'th location in P but is anywhere to the left in Q. When we relabel according to Q, this translates so we count 1 if symbol i is greater than an symbol 2^k to its right.

We shall split P' into two subsequences, one which consists only of the symbols at odd locations in P', and the other of the symbols which occur at even locations. Symbols of P' will now be referred to as 'odd symbols' or 'even symbols': this refers only to their location, not whether the value of an symbol is odd or even. Suppose s_{odd} is the length of a longest increasing subsequence of symbols at odd locations in P', and s_{even} is similarly defined for the even symbols. Define $b(P')$ as the number of locations ('sequence breakpoints') where $P'[i] > P'[i+1]$

Lemma 1. $LIS(P') \geq s_{odd} + s_{even} - b(P')$.

Proof. Let S_{even} represent an increasing sequence of even symbols whose length is s_{even}, and define S_{odd} similarly. We shall see how we can build a longer increasing subsequence starting from each of the subsequences of even and odd symbols. Consider an symbol of S_{even}, $P'[i]$ and the subsequent symbol of S_{even}, $P'[j]$. There is at least one odd symbol separating these two symbols when they occur in P'. Now, either all odd symbols that occur at locations between i and j have values between $P'[i]$ and $P'[j]$, in which case we could extend the increasing sequence S_{even} by including these symbols; or else they are all less than $P'[i]$ or greater than $P'[j]$. In either case, then there is a contribution of at least one to $b(P')$ from these intervening symbols. This allows us to conclude that from the increasing sequence S_{even}, then we can form an increasing sequence of length at least $2s_{even} - b(P')$, as there are $s_{even} - 1$ consecutive pairs of symbols from S_{even}, and in addition we can also consider the sequence before the first symbol. Similarly, from S_{odd}, we can find an increasing sequence of length at least $2s_{odd} - 1 - b(P')$. Further, depending on whether $|P'|$ is odd or even, we can always increase one of these bounds by 1, by considering the effect of the last member of S_{odd} and the subsequent even symbols if $|P'|$ is even, or the effect with the last of S_{even} and subsequent odd symbols if $|P'|$ is odd. We know that each of these generated increasing sequences of P' is of length at most $LIS(P')$ by definition of $LIS(P')$. Summing these, we find that $2s_{odd} + 2s_{even} - 2b(P') \leq 2LIS(P')$.

If we consider what $b(P')$ represents, we compare every $P'[i]$ to $P'[i+1]$ and count one for every disordered pair. This is telling us that the considered pair of symbols occur in P in the opposite order to which they occur in Q, by construction of P'. So $b(P')$ is exactly equivalent to the contribution to $D(P,Q)$ from $A_0 \cap B$. If we now split and consider P'_{odd} (and P'_{even}), the subsequences of P' formed by taking all the symbols at odd (even) locations, we note that these have exactly the same structure, and have only the self-contained comparisons of $A_1 \cap B$, $A_2 \cap B$ to $A_{\log n-1} \cap B$. We can carry on splitting each sequence recursively into odd and even sequences, until we can split no further. At the last level, all that remains are $|P'| = |P|$ single symbols, which each constitute a trivial increasing subsequence of length one. Telescoping the inequality, we find that $LIS(P') \geq |P| - b(P') - b(P'_{even}) - b(P'_{odd}) - b(P'_{odd_{odd}}) - b(P'_{odd_{even}}) \ldots$ If we sum all these b's, we get exactly $D(P,Q)$. Hence we conclude that $LCS(P,Q) = LIS(P') \geq |P| - D(P,Q)$. Rearranging and substituting, we find $D(P,Q) \geq n - LCS(P,Q) = d(P,Q)$, as required.

2.4 Combining All Operations

We consider the compound distance allowing the combination of reversals, transpositions and permutation editing (moving a single symbol). Denote this distance as $\tau(P,Q)$. We make use of the transformation R from Section 2.1, and omit the simple proof.

Theorem 4. $\tau(P,Q) \leq \frac{1}{2}H(R(P),R(Q)) \leq 3\tau(P,Q)$.

These embedding techniques can also be adapted for a large range of permutation distances. Embeddings can be obtained for variations where inserts and deletes are permitted for any of the distances already described; when one of the sequences is allowed to be a string rather than a permutation; and in the case of signed permutations. Exact details of these embeddings are omitted for brevity.

3 Implications of the Embeddings

We can immediately find algorithmic applications of our embeddings. On the whole, these rely on known results for the Hamming space.

Approximating Pairwise Distances. The embeddings allow distance approximations to be made efficiently in a communication setting. We have the following scenario: there are two communicants, A and B, who each hold a permutation P and Q respectively, and they wish to communicate in such a way to calculate the approximate distance between their permutations.

Theorem 5. *There is a single round communication protocol to allow reversal (transposition) distance approximation up to a factor of* $2+\epsilon$ *(respectively* $3+\epsilon$*) with a message of size* $O(\log n \log(1/\delta)/\epsilon^2)$*. The protocol succeeds with probability* $1-\delta$*.*

This follows from known communication results on Hamming distance such as Corollary B of [5]. Now suppose that we have a number of permutations, and we wish to be able to rapidly find the distance between any pair of them. Traditional methods would suggest that for each pair, we should take one and relabel it as the identity permutation, and then solve the sorting by reversals or sorting by transpositions problem for the correspondingly relabelled permutation. We claim that, given a near linear amount of preprocessing, this problem can be solved exponentially faster.

Corollary 1. *With a linear amount of preprocessing, the Reversal distance (respectively, Transposition distance) between any pair of permutations of length n can be approximated in time $O(\log n \log(1/\delta)/\epsilon^2)$ up to a factor of $2+\epsilon$ (resp. $3+\epsilon$).*

This follows from the above statement, since whenever we have a one round communication protocol, we can precompute and store the message that would be sent for each permutation. Pairwise approximation can then be carried out by comparing the two corresponding precomputed messages, which requires time linear in the size of the message.

Approximate Nearest Neighbors. The problem is to preprocess a collection of permutations so that given a new query permutation, the closest permutation from the collection can be found. This problem is analogous to vector nearest neighbors under Hamming metric [9,12]. The crux here is to avoid the dimensionality curse: that is, design a polynomial space data structure that answers queries in time polynomial in the query and sublinear in the collection size.

Theorem 6. *We can find approximate nearest neighbors under Reversal distance (respectively Transposition distance and compound distances thereof) up to a factor of $2+\epsilon$ (respectively $3+\epsilon$) with query time $O(\ell \cdot n^{1/(1+\epsilon)})$, where n is the number of sequences in the database, and ℓ the size of the universe from which sequence symbols are drawn.*

Proof. This follows immediately from the results for Approximate Nearest Neighbors in [9] and [12]. Some care is needed, since for efficiency we need to ensure that the sampling at the root of the Locality-Sensitive Hash functions used therein does not attempt to sample directly from the quadratic ($O(\ell^2)$) space of the matrices of the embeddings. Instead, we consider in turn each adjacent pair in a permutation, and use hash functions to determine whether this pair would have been picked by the sampling scheme.

Distance Estimation in the Streaming Model. An additional feature of the embeddings is that they lead themselves to solving problems in the streaming model.

Theorem 7. *If the sequences arrive as arbitrarily interleaved streams, approximations for the Transposition distance or Reversal distance can be computed using storage of size $O(\log \ell \log(1/\delta)/\epsilon^2)$ such that the Reversal distance (respectively Transposition distance) can be approximated to a factor of $2+\epsilon$ (resp. $3+\epsilon$) with probability $1-\delta$.*

Proof. Since each non-zero entry in the transformation matrices comes from information about adjacent pairs in the permutation, we can parse the permutation as a stream of tuples, so $\ldots i, j, k \ldots$ is viewed as $\ldots (i, j), (j, k) \ldots$. The streaming algorithm of [5] can then be used on the induced bitstring (only non-zero bits need to be handled). Although the matrix space is $O(\ell^2)$, the space needed will still be $O(\log \ell)$ in size, and can be computed with a linear pass over each permutation.

4 Approximate Permutation Matching

The counting version of Approximate Permutation Matching is stated as follows: Given a text string T of length n, and a pattern permutation P of length m, find the approximate cost of aligning the pattern against each location in the text. That is, for each i find the appropriate distance $d[i]$ between $T[i : i+m-1]$ and $P[1 : m]$. Naively using the transformations of our distances would be expensive; we take advantage of the fact that because the embeddings are based on pairwise comparisons, the approximate cost can be calculated incrementally with only a small amount of work.

Theorem 8. *i) Approximate permutation matching for reversal distance can be solved in time $O(n + m)$; each $d[i]$ is approximated to a factor of 2.*

ii) Approximate permutation matching for transposition distance can be solved in time $O(n + m)$; each $d[i]$ is approximated to a factor of 3.

Proof. We must allow insertions and deletions to our sequences since in this scenario we cannot insist that we will always find exact permutations at each alignment location. We shall make use of extended embeddings which allow these approximations to be made. It is important to note that although these embeddings are described in terms of quadratic sized matrices, we do not construct these matrices, but instead concentrate only on the linear number of non-zero entries in these figurative matrices. We shall prove both claims together, since we take advantage of common properties of the embeddings.

Suppose we know the cost of aligning $T[i \ldots i + m - 1]$ against P, and we now want to find the cost for $T[i + 1 \ldots i + m]$. This is equivalent to adding a character to the end of $T[i \ldots i + m - 1]$ and removing one from the front. So only two adjacencies are affected — at the start and end of the subsequence. This affects only a constant number of symbols in our matrices. Consequently, we need only perform a constant amount of work to update our count of the number of transposition or reversal breakpoints, provided we have precomputed the inverse of the pattern P in time $O(m)$. To begin the process, we imagine aligning P with a location to the left of T so that there is no overlap of pattern and text. Initially, the distance between P and $T[-m \ldots 0]$ is defined as m. From this position, we can advance the pattern by one location at a time and do a constant amount of work to update the count. The total time required is $O(n + m)$.

Theorem 9. *Approximate Permutation Matching can be solved for Permutation Edit Distance in time* $O(n \log m)$.

Proof. We make use of the transformation for permutation edit distance, and so our result will be accurate up to a factor of $\log m$. We can use the trick of relabelling P as $1\ 2 \ldots m$, and relabelling T accordingly as we go along. Suppose we have found the cost of matching $T[i \ldots i+m-1]$ against P. We can advance this match by one location to the right by comparing $T[i+m]$ with the $\log m$ locations $T[i+m-1], T[i+m-2], T[i+m-4] \ldots$. Each pair of the form $T[i+m-2^k] > T[i+m]$ that we find adds one to our total. At the same time, we maintain a record of the L_1 difference between the number of symbols in P missing from $T[i \ldots i+m-1]$ (since each of these must participate in an insertion operation to transform $T[i \ldots i+m-1]$ into P). This can be updated in constant time using $O(P)$ space. We can step the left end of a match by one symbol in constant time if we also keep a record for each $T[i]$ how many comparisons it caused to fail from symbols to the right — we reduce the count by this much to find the cost of $T[i+1 \ldots i+m]$ from $T[i \ldots i+m]$. In total we do $O(\log m)$ work per step, giving a total running time of $O(n \log m)$.

5 Discussion

We present the first known results embedding various permutation distances into Hamming and Set Intersection spaces in an approximately distance preserving manner. These embeddings are approximate, since finding the distances exactly is provably hard. From these embeddings, a wide variety of problems can be solved, the full extent of which is beyond the scope of this paper. In particular, we have described how the embeddings enable the solution of traditional problems such as pair-wise distance estimation and nearest neighbors; and novel problems, such as approximate permutation matching and measurements in the streaming model. These results are of interest in Computational Biology as well as for foundational reasons since analogous problems are open for strings. In solving approximate permutation matching problems, we obtained linear or near-linear time approximation algorithms while their string counterparts take significantly longer. We hope that our study of permutation distances gives insights that may help solve the corresponding open problems on string distances. A candidate problem to think about seems to be approximating the longest common subsequence, a dual of our permutation edit distances.

Acknowledgements. The first author wishes to thank Mike Paterson for some fruitful discussions about this work; in particular, for suggesting the form of Lemma 1 that enabled the proof of Theorem 3. We also thank the anonymous reviewers for their comments.

References

1. V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. In *Proceedings of the 34th Annual Symposium on Foundations of Comptuer Science*, pages 148–157, Palo Alto, CA, 1993. IEEE Computer Society Press.

2. Vineet Bafna and Pavel A. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240, May 1998.
3. A. Caprara. Sorting by reversals is difficult. In *Proceedings of the First International Conference on Computational Molecular Biology*, pages 75–83, 1997.
4. David A. Christie. A 3/2-approximation algorithm for sorting by reversals. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 244–252, San Francisco, California, 25–27 January 1998.
5. J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1-difference algorithm for massive data streams. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 501–511, 1999.
6. Vincent Ferretti, Joseph H. Nadeau, and David Sankoff. Original synteny. In *Combinatorial Pattern Matching, 7th Annual Symposium*, volume 1075 of *Lecture Notes in Computer Science*, pages 159–167. Springer, 1996.
7. Leslie Ann Goldberg, Paul W. Goldberg, Mike Paterson, Pavel Pevzner, Süleyman Cenk Şahinalp, and Elizabeth Sweedyk. The complexity of gene placement. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 386–395, N.Y., January 17–19 1999. ACM-SIAM.
8. Qian-Ping Gu, Shietung Peng, and Hal Sudborough. A 2-approximation algorithm for genome rearrangements by reversals and transpositions. *Theoretical Computer Science*, 210(2):327–339, 17 January 1999.
9. Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 604–613, 1998.
10. Howard Karloff. Fast algorithms for approximately counting mismatches. *Information Processing Letters*, 48(2):53–60, November 1993.
11. J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1/2):180–210, January 1995.
12. E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 614–623, 1998.
13. J. H. Nadeau and B. A. Taylor. Lengths of chromosome segments conserved since divergence of man and mouse. *Proc. Nat'l Acad. Sci. USA*, 81:814–818, 1984.
14. D. Sankoff and J. Nadeau. Conserved synteny as a measure of genomic distance. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 71, 1996.

Testing Hypergraph Coloring*

Artur Czumaj[1] and Christian Sohler[2]

[1] Department of Computer and Information Science, New Jersey Institute of Technology, University Heights, Newark, NJ 07102-1982, USA. czumaj@cis.njit.edu
[2] Heinz Nixdorf Institute and Department of Mathematics & Computer Science, University of Paderborn, D-33095 Paderborn, Germany. csohler@uni-paderborn.de

Abstract. In this paper we initiate the study of testing properties of hypergraphs. The goal of property testing is to distinguish between the case whether a given object has a certain property or is "far away" from the property. We prove that the fundamental problem of ℓ-colorability of k-uniform hypergraphs can be tested in time independent of the size of the hypergraph. We present a testing algorithm that examines only $(k\,\ell/\epsilon)^{\mathcal{O}(k)}$ entries of the adjacency matrix of the input hypergraph, where ϵ is a distance parameter independent of the size of the hypergraph. Notice that this algorithm tests only a constant number of entries in the adjacency matrix provided that ℓ, k, and ϵ are constant.

1 Introduction

A classical problem in computer science is to verify if a given object possesses a certain property. For example, we want to determine if a boolean formula is satisfiable, or if a graph is connected. In its very classical formulation, the goal is to give an exact solution to the problem, that is, to provide an algorithm that always returns a correct answer. In many situation, however, this formulation is too restrictive, for example, because there is no fast (or just fast enough) algorithm that gives the exact solution. Recently, many researchers started studying a relaxation of the "exact decision task" and considered various forms of approximation algorithms for decision problems. In *property testing* (see, e.g., [1,11,13, 17,16,18,19,26,29]), one considers the following class of problems:

> Let $\mathfrak{C}$ be a class of objects, $\mathfrak{O}$ be an unknown object from $\mathfrak{C}$, and $\mathcal{Q}$ be a fixed property of objects from $\mathfrak{C}$. The goal is to determine (possibly probabilistically) if $\mathfrak{O}$ has property $\mathcal{Q}$ or if it is *far* from any object in $\mathfrak{C}$ which has property $\mathcal{Q}$, where distance between two objects is measured with respect to some distribution $\mathcal{D}$ on $\mathfrak{C}$.

The motivation behind this notion of property testing is that while relaxing the exact decision task we expect the testing algorithm to be significantly more efficient than any exact decision algorithm, and in many cases, we achieve this goal by exploring only a small part of the input.

A notion of property testing was first explicitly formulated in [31] and then extended and further developed in many follow-up works (see, e.g., [1,6,7,13,

* Research supported in part by an SBR grant No. 421090 and DFG grant Me872/7-1.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 493–505, 2001.

[14,18,19,30]). Property testing arises naturally in the context of program verification, learning theory, and, in a more theoretical setting, in probabilistically checkable proofs. For example, in the context of program checking, one may first choose to test whether the program's output satisfies certain properties before checking that it is as desired. This approach is a very common practice in software development, where it is (typically) infeasible to require to formally test that a program is correct, but by verifying whether the output satisfies certain properties one can gain a reasonable confidence about the quality of the program's output.

The study of property testing for *combinatorial objects*, and mainly for labeled graphs, was initiated by Goldreich et al. [18]. They investigated several interesting graph properties and showed, for example, that testing ℓ-colorability of graphs is testable in time independent of the input size.

We refer the reader to the excellent survey by Ron [29], where a very thorough exposition of this field is presented and applications of this model are discussed.

1.1 Our Contribution

In this paper we extend the notion of property testing to hypergraphs, and study the problem of *testing colorability properties of hypergraphs.*

Hypergraphs. Recall that a *hypergraph* is a pair $\mathcal{H} = (V, E)$ such that E is a subset of the power set of V. The set V is the set of *vertices* and E is the set of *edges*. We consider only finite hypergraphs (i.e., V is finite) and such that $V \cap E = \emptyset$. If E contains only sets of size k then $\mathcal{H}$ is said to be *k-uniform*. A hypergraph is a well-known generalization of a graph; a 2-uniform hypergraph is a standard undirected graph.

An *ℓ-coloring* of a hypergraph $\mathcal{H} = (V, E)$ is a mapping $\chi : V \to \{1, \ldots, \ell\}$. An ℓ-coloring χ of a hypergraph $\mathcal{H} = (V, E)$ is called *proper* if $\mathcal{H}$ contains no monochromatic edge (that is, for every $e \in E$, there exist $x, y \in e$ such that $\chi(x) \neq \chi(y)$). A hypergraph $\mathcal{H}$ is ℓ-colorable, if there exists a proper ℓ-coloring of $\mathcal{H}$.

In the case when we will discuss the 2-coloring problem, we shall frequently consider χ to be a function that assigns to every vertex either color *red* or *blue*.

Testing colorability property of hypergraphs. In this paper we study the problem of testing the property that a given hypergraph is ℓ-colorable. We assume the hypergraph $\mathcal{H} = (V, E)$ with n vertices is k-uniform and it is represented by its *adjacency matrix* A of size n^k, that is, the entry $A[v_{i_1}, v_{i_2}, \ldots, v_{i_k}]$ is equal to 1 if and only if $\{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\} \in E$; it is 0 otherwise.

In general case, we are using the following definition:

Definition 1.1. *Let $\mathcal{P}$ be any property of hypergraphs. Let ϵ be any real $0 \leq \epsilon \leq 1$. A k-uniform hypergraph $\mathcal{H} = (V, E)$ is* ϵ-far from property $\mathcal{P}$ *if it has Hamming distance at least ϵn^k from any hypergraph having property $\mathcal{P}$, that is, in order to construct from $\mathcal{H}$ a hypergraph having property $\mathcal{P}$ one has to delete or insert at least ϵn^k edges of $\mathcal{H}$.*

Using this notion of distance, we can formally define testing algorithms:

Definition 1.2. *Let $\mathcal{P}$ be any property of hypergraphs. Let ϵ be any real $0 \leq \epsilon \leq 1$. An* ϵ-tester *for property $\mathcal{P}$ of k-uniform hypergraphs is an algorithm that*

- accepts *every hypergraph having property $\mathcal{P}$, and*
- rejects *with probability at least $2/3$ any hypergraph that is ϵ-far from property $\mathcal{P}$.*

Observe that the behavior of an *ϵ-tester* may be arbitrary for hypergraphs that neither have property $\mathcal{P}$ nor are ϵ-far from property $\mathcal{P}$.

Specifically, given query access to an adjacency matrix A representing $\mathcal{H}$, and a distance parameter ϵ, we study the problem of determining with reasonably high probability whether $\mathcal{H}$ is ℓ-colorable, or whether more than an ϵ-fraction of entries of A should be modified so that the hypergraph defined by the modified adjacency matrix becomes ℓ-colorable. In the later case, we say $\mathcal{H}$ is ϵ-far from being ℓ-colorable.

There are two measures of the complexity of testing algorithms: the *query complexity* and the *running time complexity* of an ϵ-tester. The query complexity of a tester (in our context of hypergraphs) is measured only by the number of queries to the entries of the adjacency matrix of the input hypergraph, while the running time complexity counts also the time needed by the algorithm to perform other tasks (e.g., to verify if a given sub-hypergraphs is ℓ-colorable).

To exemplify the notion of ϵ-testers, let us compare the notion of standard approximation of 2-colorability with the notion of testing 2-colorability in 3-uniform hypergraphs (this is a slight modification of an example used in [7]):

> *A hypergraph $\mathcal{H}$ might be* nearly 2*-colorable in the sense that there is a 2-colorable hypergraph $\mathcal{H}^*$ at small Hamming distance to $\mathcal{H}$, but* far *from 2-colorable in the sense that many colors are required to properly color $\mathcal{H}$. Similarly, a hypergraph $\mathcal{H}$ might be* nearly 2*-colorable in the sense that it is 3-colorable, but far from 2-colorable in the sense that no hypergraphs having small Hamming distance to $\mathcal{H}$ are 2-colorable. Therefore, both these notions are natural and the preferred choice depends on the application at hand.*

Results. Our main theorem is an ϵ-tester for ℓ-colorability of k-uniform hypergraphs that has query complexity that is independent of the input hypergraph size.

Our ϵ-tester follows the standard approach in this area: it first samples at random a subset of vertices of the hypergraph $\mathcal{H}$, and then checks whether the sub-hypergraph of $\mathcal{H}$ induced by the vertices chosen is colorable:

> Tester(s, ℓ)
>
> *Pick a subset $\mathcal{S} \subseteq V$ of size s uniformly at random.*
>
> *Let $\mathcal{H}_{\mathcal{S}}$ be the hypergraph induced by $\mathcal{S}$ in $\mathcal{H}$.*
>
> **If** *$\mathcal{H}_{\mathcal{S}}$ is ℓ-colorable* **then** accept $\mathcal{H}$*;*
>
> **else** reject $\mathcal{H}$.

We can prove the following result.

Theorem 1.1. $\mathsf{Tester}(s,\ell)$ *with* $s = \tilde{\mathcal{O}}((k\,\ell/\epsilon)^2)$ *is an* ϵ*-tester for* ℓ*-coloring* k*-uniform hypergraphs.*[1]

This immediately implies the following.

Theorem 1.2. *There is an* ϵ*-tester for* ℓ*-colorability of* k*-uniform hypergraphs that has query complexity* $\tilde{\mathcal{O}}((k\,\ell/\epsilon)^{2\,k})$ *and the running time of* $\exp(\tilde{\mathcal{O}}(k\,\ell/\epsilon)^2)$.

1.2 Context and Related Work

Hypergraph coloring. Hypergraph coloring is a well studied problem in the literature in discrete mathematics, combinatorics, and computer science. In contrast to graphs, where one can decide in linear time if a graph is 2-colorable (or equivalently, bipartite), testing if a given hypergraph is 2-colorable is $\mathcal{NP}$-hard even for 3-uniform hypergraphs [23]. In [22], it is shown that unless $\mathcal{NP} \subseteq \mathcal{ZPP}$, for any fixed $k \geq 3$, it is impossible to approximate in polynomial time the chromatic number of k-uniform hypergraphs within a factor $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$. Very recently, Guruswami et al. [20] proved that for any constant c, it is $\mathcal{NP}$-hard to color a 2-colorable 4-uniform hypergraph using c colors. In [20] even a stronger inapproximability result is shown, that there exists a constant c_0 such that, unless $\mathcal{NP} \subseteq DTIME(n^{\mathcal{O}(\log\log n)})$, there is no polynomial time algorithm that colors a 2-colorable 4-uniform hypergraph using $c_0 \log\log\log n$ colors.

The property of hypergraph 2-colorability (called also "Property B" by Erdős) has been extensively studied in the combinatorics literature (see, e.g., [5,10,12,27]). In particular, the study of this problem led to the discovery of the celebrated Lovász Local Lemma [12]. In computer science the problems of coloring hypergraphs have been studied mostly due to its connection to important graph coloring and satisfiability problems (cf., e.g., [9,24]). Extending the approximation results for graph coloring, several authors have provided approximation algorithms for coloring 2-colorable hypergraphs [2,8,21,22]. For example, the very recent polynomial-time approximation algorithm from [21] colors any 3-uniform 2-colorable hypergraphs using $\tilde{\mathcal{O}}(n^{1/5})$ colors.

Testing colorability. We are not aware of any prior testing algorithms for colorability of hypergraphs. However, very recently we have heard that, independently to our work, Alon and Shapira (personal communication, 2001) developed a testing algorithm for some general version of satisfiability that includes also testing ℓ-colorability of uniform hypergraphs.

Goldreich et al. [18] were the first who studied the problem of testing ℓ-colorability in graphs (although implicitly this problem could be traced to [28]). In the most basic case of graph 2-coloring (that is, testing bipartitness), they designed an algorithm with $\tilde{\mathcal{O}}(1/\epsilon^3)$ query complexity (and running time). Their analysis was later improved by Alon and Krivelevich [3], who showed that the

[1] $\tilde{\mathcal{O}}$ is a standard asymptotic notation that "hides" polylogarithmic factors.

complexity of this algorithm is $\tilde{\mathcal{O}}(1/\epsilon^2)$. For the more general case of testing ℓ-colorability for arbitrary $\ell \geq 2$, Goldreich et al. [18] presented an algorithm with the query complexity of $\tilde{\mathcal{O}}(\ell^4/\epsilon^6)$ and the running-time complexity of $2^{\tilde{\mathcal{O}}(\ell^2/\epsilon^3)}$. Again, Alon and Krivelevich [3] improved the analysis of the algorithm and obtained a bound of $\tilde{\mathcal{O}}(\ell^2/\epsilon^4)$ on the query complexity and $2^{\tilde{\mathcal{O}}(\ell/\epsilon^2)}$ on the running time. Alon et al. [1] presented another "constant-time" (i.e., independent of the size of the input graph) property testing algorithm; their algorithm uses the Szemerédi Regularity Lemma, and therefore the bounds for the query complexity and the running time, though independent of the size of the graph, have huge dependency of ℓ and ϵ. Fischer [15] extended the methods from [1] and investigated more general graph colorability properties.

1.3 Organization of the Paper

Because of space limitations, we concentrate our analysis mostly on testing 2-colorability of 3-uniform hypergraphs and only briefly discuss extensions to the general case. In the main part of the paper, in Section 2, we present a detailed analysis of $\mathsf{Tester}(s, 2)$ and prove Theorems 1.1 and 1.2 for 2-colorability of 3-uniform hypergraphs. Then, in Section 3, we briefly discuss extensions of this result to ℓ-colorability of k-uniform hypergraphs.

2 Testing 2-Colorability of 3-Uniform Hypergraphs

In this section we only consider 2-coloring of 3-uniform hypergraphs. Let $\mathcal{H} = (V, E)$ be a 3-uniform hypergraph. This section is devoted to the proof the following result.

Theorem 2.1. $\mathsf{Tester}(s, 2)$ *with* $s = \mathcal{O}((1/\epsilon)^2)$ *is an* ϵ*-tester for* 2*-coloring* 3*-uniform hypergraphs.*

Theorem 2.1 immediately implies the following.

Theorem 2.2. *There is an* ϵ*-tester for* 2*-coloring* 3*-uniform hypergraphs with query complexity of* $\Theta(1/\epsilon^6)$ *and the running time of* $\exp(\mathcal{O}(1/\epsilon^2))$. □

We choose $s = 4 \cdot 10^3 \cdot (1/\epsilon)^2$, though we did not try to optimize the constant and it is easy to improve over our constant $4 \cdot 10^3$ significantly, perhaps even to a one digit number.

In order to prove Theorem 2.1 we must show the following properties of $\mathsf{Tester}(s, 2)$:

1. if $\mathcal{H}$ is 2-colorable, then the algorithm accepts $\mathcal{H}$ (that is, $\mathcal{H}_\mathcal{S}$ is 2-colorable);
2. if $\mathcal{H}$ is ϵ-far from 2-colorable, then the algorithm rejects $\mathcal{H}$ (that is, $\mathcal{H}_\mathcal{S}$ is not 2-colorable) with probability at least 2/3.

Since if a hypergraph is 2-colorable, so is any its sub-hypergraph (and in particular, $\mathcal{H}_\mathcal{S}$), property (1) trivially holds. Therefore we must only prove that property (2) holds as well. From now on, we shall assume $\mathcal{H}$ is ϵ-far from having 2-coloring.

2.1 Coloring Game with the Adversary

For the purpose of the analysis, we partition our sample set $\mathcal{S}$ into $100/\epsilon$ sets U_i, $1 \leq i \leq 100/\epsilon$, of size $40/\epsilon$ each.

We analyze the following *game* on $\mathcal{H}$:

We play $100/\epsilon$ rounds starting with an initially empty set $V_{colored}$ of *colored* vertices. In the course of the game we are adding new vertices to $V_{colored}$ and the *adversary* chooses a color for each of these vertices. The coloring procedure of the adversary may be arbitrary, but the partial coloring of $\mathcal{H}$ on the sub-hypergraph induced by $V_{colored}$ must be always proper. If the adversary is unable to properly color the vertex chosen, then *we win*. If the adversary properly colors the vertices during all $100/\epsilon$ rounds, *he wins*.

Formally, round i of the game looks as follows:

- We choose a vertex v from set U_i and add it to $V_{colored}$.
- The adversary colors v either red or blue. He is not allowed to create monochromatic edges.

The following claim that plays the key role in our analysis explains the idea behind introducing the game.

Claim. If for any 3-uniform hypergraph $\mathcal{H}$ that is ϵ-far from 2-colorable we win independently of the strategy of the adversary with probability at least 2/3, then the hypergraph $\mathcal{H}_{\mathcal{S}}$ computed by $\mathsf{Tester}(s, 2)$ is not 2-colorable with probability at least 2/3. Therefore, in particular, $\mathsf{Tester}(s, 2)$ is an ϵ-tester for 2-coloring 3-uniform hypergraphs.

Proof. The proof is by contradiction. Let us assume that $\mathcal{H}_{\mathcal{S}}$ has a proper coloring $\chi_{\mathcal{H}_{\mathcal{S}}}$ with probability greater than 1/3 (over the choice of $\mathcal{S}$). Then, the adversary may color each vertex $v \in \mathcal{S}$ according to $\chi_{\mathcal{H}_{\mathcal{S}}}(v)$. Since the adversary wins if $\chi_{\mathcal{H}_S}$ is proper, he wins with probability greater than 1/3, which is a contradiction.

By our discussion above, this implies that $\mathsf{Tester}(s, 2)$ is an ϵ-tester for 2-coloring 3-uniform hypergraphs. □

Therefore, our plan is to show that if $\mathcal{H}$ is ϵ-far from 2-colorable, then we win the game with probability at least 2/3 independently of the strategy of the adversary. In order to prove this result, we first concentrate ourself on estimating the probability that we win against a single fixed strategy of the adversary, and then generalize this estimation to winning against all strategies of the adversary.

2.2 Our Strategy

Informally, our strategy in round i is to choose an especially selected vertex v from U_i that either cannot be properly colored or that adds many new "constraints" to the colors of the vertices of the hypergraph no matter what color the adversary chooses to color v.

During the game, some of the vertices are already colored. This coloring defines *constraints* for the colors of the remaining, yet uncolored vertices. We model these constraints by five sets $V_{colored}$, $V_{conflict}$, V_{red}, V_{blue}, V_{free} that form a partition of the vertex set V, and by two graphs $G_{red} = (V, E_{red})$ and $G_{blue} = (V, E_{blue})$.

$V_{colored}$**:** contains all vertices that have been already colored by the adversary.
$V_{conflict}$**:** contains all yet uncolored vertices that are incident to both an edge with two blue vertices (in $V_{colored}$) and another edge with two red vertices (in $V_{colored}$); notice that these vertices cannot be properly colored by the adversary.
V_{red}**:** contains all yet uncolored vertices that are not in $V_{conflict}$ and can be properly colored only in red (that is, these are vertices incident to an edge with two blue vertices in $V_{colored}$).
V_{blue}**:** contains all yet uncolored vertices that are not in $V_{conflict}$ can be properly colored only in blue (that is, these are vertices incident to an edge with two red vertices in $V_{colored}$).
V_{free}**:** contains all remaining vertices (that is, yet uncolored vertices that can be properly colored both red and blue).
G_{red}**:** contains an edge between two vertices v and w in V, if and only if there is an edge $e = \{v, w, u\}$ with a red colored vertex $u \in V_{colored}$ (thus, an edge in G_{red} means that coloring both its endpoints red creates a monochromatic edge).
G_{blue}**:** contains an edge between two vertices v and w in V, if and only if there is an edge $e = \{v, w, u\}$ with a blue colored vertex $u \in V_{colored}$ (thus, an edge in G_{blue} means that coloring both its endpoints blue creates a monochromatic edge).

Now, in order to formalize our strategy we define *heavy vertices.*

Definition 2.1. *Let $\mathcal{H} = (V, E)$ be a 3-uniform hypergraph. Let $V_{colored}$ be a subset of V that is properly 2-colored by $\chi : V_{colored} \to \{red, blue\}$. A vertex $v \in V - V_{colored}$ is called* heavy *for $(V_{colored}, \chi)$ if at least one of the following two conditions is satisfied after extending χ by any proper coloring of v:*

- *there are at least $\epsilon n^2/10$ new edges between vertices either in G_{red} or in G_{blue}, or*
- *there are at least $\epsilon n/10$ new vertices in one of the sets V_{red} , V_{blue} , or $V_{conflict}$.*

Now, we state our main lemma about heavy vertices:

Lemma 2.1. *Let $\mathcal{H} = (V, E)$ be a 3-uniform hypergraph and let $V_{colored}$ be an arbitrary subset of its vertices that is properly 2-colored by $\chi : V_{colored} \to \{red, blue\}$. Then, one of the following conditions hold:*

- *$\mathcal{H}$ is* not *ϵ-far from 2-colorable,*
- *there are at least $\epsilon n/10$ heavy vertices for $(V_{colored}, \chi)$,*
- *$|V_{conflict}| \geq \epsilon n/10$.*

Proof. The proof is by contradiction. Suppose none of the three conditions above holds. Then, $\mathcal{H}$ is ϵ-far from 2-colorable. Using the negation of the other two conditions we will construct a 2-coloring of $\mathcal{H}$ that violates less than ϵn^3 edges. This implies that $\mathcal{H}$ is not ϵ-far from 2-colorable, which is a contradiction.

The algorithm below constructs a 2-colorable hypergraph $\mathcal{H}'$ by deleting less than ϵn^3 edges from $\mathcal{H}$.

At the beginning of the algorithm we fix the sets V_{red}, V_{blue}, V_{free}, and $V_{conflict}$ as well as the graphs G_{red} and G_{blue}. Then the algorithm colors the vertices one after the other. Each time a vertex is colored its coloring may introduce new constraints, that is, new vertices in the sets V_{red}, V_{blue}, or $V_{conflict}$ or new edges in the graphs G_{red} or G_{blue}. For each such new constraint there is a set of edges that is responsible for the new constraint. These edges are called the witnesses of the new constraint. E.g., if vertex v is colored red, then the edge $\{v, u, w\}$ is a witness for the edge (constraint) (u, w) in G_{red}. The algorithm deletes all witnesses for new constraints. Thus, it maintains the following invariant at the beginning of each **for each** loop:

> *The constraints for the colors of uncolored vertices are given by (a subset of) the constraints represented by the sets V_{red}, V_{blue}, V_{free}, $V_{conflict}$, and the graph G_{red} and G_{blue}. E.g., if a vertex is in the set V_{red} it can be colored red without creating monochromatic edges in the current hypergraph at any time in the algorithm.*

Below it is proven that we can maintain this invariant by removing less than ϵn^3 edges.

```
ConstructColoring(H)
    for each v ∈ V that is either heavy or is in V_conflict do
        χ(v) = red
        remove all edges incident to v
    for each v ∈ V_red that is not heavy do
        χ(v) = red
        remove all edges that cause new constraints
    for each v ∈ V_blue that is not heavy do
        χ(v) = blue
        remove all edges that cause new constraints
    for each v ∈ V_free that is not heavy do
        if coloring v red causes fewer new constraints than coloring v blue then
                χ(v) = red
        else
                χ(v) = blue
        remove all edges that cause new constraints
```

In what follows we prove that the so obtained hypergraph $\mathcal{H}'$ is properly 2-colored by χ and that it is obtained from $\mathcal{H}$ by deleting less than ϵn^3 edges. It is easy to see that the algorithm maintains the invariant that the constraints for the colors of the remaining vertices do not change. Indeed, if coloring a certain vertex creates new constraints, then all edges that cause these constraints are

deleted from the hypergraph. Thus at any time, coloring a vertex in V_{red} (V_{blue}) red (blue) does not create any monochromatic edges in the current hypergraph. Coloring heavy and conflict vertices obviously is correct because all incident edges are deleted. And finally, coloring a vertex in V_{free} either red or blue again does not create any monochromatic edges because of the invariant. Therefore, the obtained hypergraph $\mathcal{H}'$ is properly 2-colored by χ.

It remains to show that the number of deleted edges is less than $\epsilon\, n^3$.

We remove at most n^2 edges incident to any heavy vertex or a vertex in $V_{conflict}$. Since we know that there are less than $\epsilon\, n/10$ heavy vertices as well as less than $\epsilon\, n/10$ vertices in $V_{conflict}$, the loop over these two sets of vertices (that removes all incident edges) will delete less than $2\,\epsilon\, n^3/10$ edges.

All remaining vertices are not heavy. Thus, coloring any such a vertex will create less than $\epsilon\, n/10$ new constraints in V_{red}, V_{blue}, and $V_{conflict}$ and less than $\epsilon\, n^2/10$ new constraints in G_{red} and G_{blue} (cf. Definition 2.1). Each of the new constraints in V_{red}, V_{blue}, and $V_{conflict}$ can cause at most n edges to become new constraints. Since there are at most n vertices in $V_{red} \cup V_{blue} \cup V_{free}$, the last three loops delete at most $5\,\epsilon\, n^3/10$ edges from $\mathcal{H}$.

Thus, overall, the hypergraph $\mathcal{H}'$ is obtained from $\mathcal{H}$ by deleting less than $7\,\epsilon\, n^3/10$ edges. This yields a contradiction, because on one hand we have assumed that $\mathcal{H}$ is ϵ-far from 2-colorable, but on the other hand we have just shown that there is a 2-colorable hypergraph $\mathcal{H}'$ that is obtained from $\mathcal{H}$ by deletion of less than $\epsilon\, n^3$ edges. □

2.3 Proof of Theorem 2.1

Now we are ready to formulate our strategy in details and to complete the proof of Theorem 2.1. We consider only the case that $\mathcal{H}$ is ϵ-far from 2-colorable. We want to show that for any strategy of the adversary, we win with probability at least 2/3. Then, Claim 2.1 would imply the proof of Theorem 2.1.

Observe that there are at most $2^{100/\epsilon}$ strategies of the adversary, each one corresponding to a binary string of length $100/\epsilon$ such that if the ith bit is 1 (or 0, respectively), then the adversary colors vertex $v \in U_i$ red (or blue, respectively). Let us fix any strategy of the adversary Υ. Then, in round i we may assume we know the current status of the game (the coloring of the vertices in P chosen prior to round i). We further may assume that the set U_i is chosen at random. Then we choose the next vertex $v \in U_i$ to be colored by the adversary as follows: If there is a vertex in U_i that belongs also to $V_{conflict}$ then we choose one such a vertex and win the game. If there is no vertex in $U_i \cap V_{conflict}$, then we choose a heavy vertex if one exists in U_i. If there is no heavy vertex in U_i, then we choose an arbitrary vertex from U_i.

Now, let us observe that since U_i is a randomly chosen set of vertices of size $40/\epsilon$, from Lemma 2.1 we may conclude that in round i

$$\mathbf{Pr}\big[v \text{ is neither heavy nor belongs to } V_{conflict} \mid \Upsilon\big] \leq (1-\epsilon/10)^{40/\epsilon} \leq e^{-4}\ . \tag{1}$$

Now, let us recall that the coloring by the adversary of any heavy vertex either inserts at least $\epsilon n^2/10$ new edges to one of G_{red} or G_{blue}, or inserts at least $\epsilon n/10$ new vertices to one of the sets V_{red}, V_{blue}, or $V_{conflict}$. Furthermore, if a vertex v is chosen that is neither heavy nor belongs to $V_{conflict}$, then the number of constraints does not decreases. Therefore, since each of the sets V_{red}, V_{blue}, or $V_{conflict}$ may have at most n vertices, and each of the graphs G_{red} or G_{blue} may have at most n^2 edges, we can conclude that a heavy vertex may be chosen at most $50/\epsilon$ times.

For a given strategy of the adversary Υ and for a given round i, $1 \leq i \leq 100/\epsilon$, let $\mathcal{X}_i^{\Upsilon}$ be the indicator random variable of the event that for the strategy of the adversary Υ (1) we have neither won in round $j < i$, (2) nor the vertex v chosen in round i either is heavy or belongs to $V_{conflict}$. Let $\mathcal{X}^{\Upsilon} = \sum_{i=1}^{100/\epsilon} \mathcal{X}_i^{\Upsilon}$. Observe that by our arguments above, if $\mathcal{X}^{\Upsilon} < 50/\epsilon$, then we win! Therefore, our goal now is to estimate the probability that $\mathcal{X}^{\Upsilon} \geq 50/\epsilon$.

By (1), for every Υ and every i, we have $\mathbf{Pr}[\mathcal{X}_i^{\Upsilon} = 1 \mid \Upsilon] \leq e^{-4}$. Therefore, we can conclude that for every Υ and every $t \in \mathbb{R}$ it holds that[2] $\mathbf{Pr}[\mathcal{X}^{\Upsilon} \geq t] \leq \mathbf{Pr}[\mathbb{B}(100/\epsilon, e^{-4}) \geq t]$, where $\mathbb{B}(N,p)$ is a binomially distributed random variable with parameters N and p, that is, $\mathbf{Pr}[\mathbb{B}(N,p) = k] = \binom{N}{k} p^k (1-p)^{N-k}$ for every $0 \leq k \leq N$. Given this majorization result, we can use basic calculations to estimate the probability that $\mathcal{X}^{\Upsilon} \geq 50/\epsilon$. Let $N = 100/\epsilon$ and $p = e^{-4}$.

$$\begin{aligned}
\mathbf{Pr}[\mathcal{X}^{\Upsilon} \geq 50/\epsilon] \leq \mathbf{Pr}[\mathbb{B}(N,p) \geq N/2] &= \sum_{k=N/2}^{N} \binom{N}{k} \cdot p^k \cdot (1-p)^{N-k} \\
&\leq \sum_{k=N/2}^{N} \left(\frac{e\,N}{k}\right)^k \cdot p^k = \sum_{k=N/2}^{N} \left(\frac{e\,N\,p}{k}\right)^k \leq \sum_{k=N/2}^{N} (2\,e\,p)^k \\
&\leq \sum_{k \leq N/2} (2\,e\,p)^k = \frac{(2\,e\,p)^{N/2}}{1-2\,e\,p} = \frac{(2/e^3)^{50/\epsilon}}{1-2/e^3} \leq \frac{1}{3} \cdot 2^{-100/\epsilon} \ .
\end{aligned}$$

Thus, we have shown that for a given strategy Υ the adversary wins with probability upper bounded by $(1/3) \cdot 2^{-100/\epsilon}$. Now, we can incorporate the union bound to obtain an upper bound for the probability that there is a strategy Υ for which the adversary wins:

$$\mathbf{Pr}\left[\exists \Upsilon \ \mathcal{X}^{\Upsilon} \geq 50/\epsilon\right] \leq \sum_{\Upsilon} \mathbf{Pr}[\mathcal{X}^{\Upsilon} \geq 50/\epsilon] \leq 2^{100/\epsilon} \cdot ((1/3) \cdot 2^{-100/\epsilon}) \leq 1/3 \ .$$

Hence, we have proven that we win for all strategies with probability greater than or equal to 2/3. By Claim 2.1, this implies the proof of Theorem 2.1. □

3 Testing ℓ-Colorability of k-Uniform Hypergraphs

In this section we briefly describe how to generalize the result from Section 2 to ℓ-colorability of k-uniform hypergraphs and prove Theorem 1.1. Our analysis

[2] This is a standard fact on majorization in probability theory, see, e.g., [4, Lemma 3.1].

follows roughly the same approach as the proof of Theorem 2.1 and we will frequently refer to that proof for some details.

Let us fix $s = 1600\,k^2\,\ell^2\,\ln \ell/\epsilon^2$ and consider $\mathsf{Tester}(s,\ell)$. Since it is easy to see that any ℓ-colorable hypergraph is accepted by the tester, it is sufficient to prove that any hypergraph that is ϵ-far from ℓ-colorable is rejected by $\mathsf{Tester}(s,\ell)$ with probability at least 2/3.

Our goal is to show that we win the game against the adversary who is now allowed to use ℓ colors instead of 2 as in Section 2. We partition the sample set $\mathcal{S}$ into $20\,k^2\,\ell^2/\epsilon$ sets $\mathcal{U}_i$, $1 \le i \le 20\,k^2\,\ell^2/\epsilon$ of size $80\,\ln \ell/\epsilon$ each.

We obtain the general result by adjusting our constraint modeling from Section 2 to ℓ-coloring of k-uniform hypergraphs. We model the constraints by a set of ℓ j-uniform hypergraphs $H_{i,j}$ for each $1 \le i \le \ell$ and $1 \le j \le k-1$. The $H_{i,2}$ are graphs and the $H_{i,1}$ are sets. Again, we also have the sets $V_{colored}$, and $V_{conflict}$.

$H_{i,j}$ contains an edge between vertices $v_1, \ldots, v_j$, if and only if there is an edge $\{v_1, \ldots, v_j, v_{j+1}, \ldots, v_k\}$ in $\mathcal{H}$ such that $v_{j+1}, \ldots, v_k$ are colored with color i. Thus an edge $\{v_1, \ldots, v_j\}$ in the hypergraph $H_{i,j}$ means that coloring vertices $v_1, \ldots, v_j$ with color i will create a monochromatic edge. Also, note that the meaning of the sets $H_{i,1}$ is different from the meaning of V_{red} and V_{blue} in Section 2 in the sense that $H_{i,1}$ contains all vertices that may *not* be colored with color i.

Definition 3.1. *Let $\mathcal{H} = (V,E)$ be a k-uniform hypergraph. Let $V_{colored}$ be a subset of V that is properly l-colored by $\chi : V_{colored} \to \{1,\ldots,\ell\}$. A vertex $v \in V - V_{colored}$ is called* heavy *for $(V_{colored},\chi)$ if at least one of the following two conditions is satisfied after extending χ by any proper coloring of v:*

- *there are at least $\epsilon\, n^j/(10\,k\,\ell)$ new edges between vertices in $H_{i,j}$ for some i,j*
- *there are at least $\epsilon\, n/10$ new vertices in the set $V_{conflict}$.*

Using similar arguments (though technically more involved) as those used in Section 2, we can prove the following main technical result.

Lemma 3.1. *Let $\mathcal{H} = (V,E)$ be a k-uniform hypergraph and let $V_{colored}$ be a subset of its vertices that is properly ℓ-colored by $\chi : V_{colored} \to \{1,\ldots,\ell\}$. Then, one of the following conditions hold:*

- *there are at least $\epsilon\, n/10$ heavy vertices for $(V_{colored},\chi)$,*
- *$|V_{conflict}| \ge \epsilon\, n/10$,*
- *$\mathcal{H}$ is* not *ϵ-far from ℓ-colorable.* □

Once we have Lemma 3.1, we can proceed similarly as in Subsection 2.3 to prove that we win the game with probability greater than or equal to 2/3 no matter which strategy is chosen by the adversary.

This implies the proof of Theorem 1.1. □

References

1. N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. In *Proc. 40th FOCS*, pages 656–666, 1999.
2. N. Alon, P. Kelsen, S. Mahajan, and H. Ramesh. Coloring 2-colorable hypergraphs with a sublinear number of colors. *Nordic Journal of Computing*, 3:425–439, 1996.
3. N. Alon and M. Krivelevich. To appear in *SIAM Journal on Discrete Mathematics*.
4. Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, September 1999.
5. J. Beck. An algorithmic approach to the Lovász local lemma. I. *Random Structures and Algorithms*, 2(4):343–365, 1991.
6. M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, December 1993.
7. M. A. Bender and D. Ron. Testing acyclity of directed graphs in sublinear time. In *Proc. 27th ICALP*, pages 809–820, 2000.
8. H. Chen and A. Frieze. Coloring bipartite hypergraphs. In *Proc. 5th IPCO*, pages 345–358, 1996.
9. A. Czumaj and C. Scheideler. An algorithmic approach to the general Lovász Local Lemma with applications to scheduling and satisfiability problems. In *Proc. 32nd STOC*, pages 38–47, 2000.
10. A. Czumaj and C. Scheideler. Coloring non-uniform hypergraphs: A new algorithmic approach to the general Lovász Local Lemma. In *Proc. 11th SODA*, pages 30–39, 2000.
11. A. Czumaj, C. Sohler, and M. Ziegler. Property testing in computational geometry. In *Proc. 8th ESA*, pages 155–166, 2000.
12. P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In A. Hajnal, R. Rado, and V. T. Sós, editors, *Infinite and Finite Sets (to Paul Erdős on his 60th birthday)*, volume II, pages 609–627. North-Holland, Amsterdam, 1975.
13. F. Ergün, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *Journal of Computer and System Sciences*, 60:717–751, 2000.
14. F. Ergün, S. Ravi Kumar, and R. Rubinfeld. Approximate checking of polynomials and functional equations. In *Proc. 37th FOCS*, pages 592–601, 1996.
15. E. Fischer. Testing graphs for colorability properties. In *Proc. 12th SODA*, pages 873–882, 2001.
16. E. Fischer and I. Newman. Testing of matrix properties. To appear in *Proc. 33rd STOC*, 2001.
17. A. Frieze and R. Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19:175–220, 1999.
18. O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, July 1998.
19. O. Goldreich and D. Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.
20. V. Guruswami, J. Håstad, and M. Sudan. Hardness of approximate hypergraph coloring. In *Proc. 41st FOCS*, pages 149–158, 2000.
21. M. Krivelevich, R. Nathaniel, and B. Sudakov. Approximating coloring and maximum independent sets in 3-uniform hypergraphs. In *Proc. 12th SODA*, pages 327–328, 2001.

22. M. Krivelevich and B. Sudakov. Approximate coloring of uniform hypergraphs. In *Proc. 6th ESA*, pages 477–489, 1998.
23. L. Lovász. Coverings and colorings of hypergraphs. In *Proc. 4th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 3–12. 1973.
24. C-J. Lu. Deterministic hypergraph coloring and its applications. In *Proc. 2nd RANDOM*, pages 35–46, 1998.
25. I. Newman. Testing of function that have small width branching programs. In *Proc. 41st FOCS*, pages 251–258, 2000.
26. M. Parnas and D. Ron. Testing metric properties. To appear in *Proc. 33rd STOC*, 2001.
27. J. Radhakrishnan and A. Srinivasan. Improved bounds and algorithms for hypergraph two-coloring. In *Proc. 39th FOCS*, pages 684–693, 1998.
28. V. Rödl and R. A. Duke. On graphs with small subgraphs of large chromatic number. *Graphs and Combinatorics*, 1:91–96, 1985.
29. D. Ron. Property testing. To appear in P. M. Pardalos, S. Rajasekaran, J. Reif, and J. D. P. Rolim, editors, *Handobook of Randomized Algorithms*. Kluwer Academic Publishers, 2001.
30. R. Rubinfeld. Robust functional equations and their applications to program testing. In *Proc. 35th FOCS*, pages 288–299, 1994.
31. R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, April 1996.

Total Colorings of Degenerated Graphs

Shuji Isobe, Xiao Zhou, and Takao Nishizeki

Graduate School of Information Sciences, Tohoku University,
Aoba-yama 05, Sendai, 980-8579, Japan.
{iso@nishizeki.,zhou@,nishi@}ecei.tohoku.ac.jp

Abstract. A total coloring of a graph G is a coloring of all elements of G, i.e. vertices and edges, in such a way that no two adjacent or incident elements receive the same color. A graph G is s-degenerated for a positive integer s if G can be reduced to a trivial graph by successive removal of vertices with degree $\leq s$. We prove that an s-degenerated graph G has a total coloring with $\Delta + 1$ colors if the maximum degree Δ of G is sufficiently large, say $\Delta \geq 4s + 3$. Our proof yields an efficient algorithm to find such a total coloring. We also give a linear-time algorithm to find a total coloring of a graph G with the minimum number of colors if G is a partial k-tree, i.e. the tree-width of G is bounded by a fixed integer k.

1 Introduction

We deal with a total coloring of a *simple graph* G, which has no multiple edges or selfloops. A total coloring is a mixture of ordinary vertex-coloring and edge-coloring. That is, a *total coloring* of G is an assignment of colors to its vertices and edges so that no two adjacent vertices have the same color, no two adjacent edges have the same color, and no edge has the same color as one of its ends [14]. The minimum number of colors required for a total coloring of a graph G is called the *total chromatic number* of G, and denoted by $\chi_{\mathrm{t}}(G)$. Figure 1(a) illustrates a total coloring of a graph G using $\chi_{\mathrm{t}}(G) = 4$ colors. Let $\Delta(G)$ be the maximum degree of G, then clearly $\Delta(G) + 1 \leq \chi_{\mathrm{t}}(G)$, and hence $\Delta(G) + 1$ is a lower bound on $\chi_{\mathrm{t}}(G)$. On the other hand, it is conjectured that an upper bound $\chi_{\mathrm{t}}(G) \leq \Delta(G) + 2$ holds for any simple graph G. However, this "total coloring conjecture" has not been verified [8,14]. The *total coloring problem* is to find a total coloring of a given graph G with the minimum number $\chi_{\mathrm{t}}(G)$ of colors. Since the problem is NP-hard [11], it is very unlikely that there exists an efficient algorithm to solve the problem for general graphs. However, there would exist an efficient algorithm for a restricted class of graphs such as "s-degenerated graphs" and "partial k-trees" defined below.

A graph is said to be *s-degenerated* for an integer $s \geq 1$ if it can be reduced to a trivial graph by successive removal of vertices with degree $\leq s$. For example, the graph in Fig. 1(a) is 2-degenerated, and every planar graph is 5-degenerated. An s-degenerated graph has a favorable property on the vertex-coloring: let $\chi(G)$ be the *chromatic number* of a graph G, that is, the minimum number of colors required for a vertex-coloring of G, then clearly $\chi(G) \leq s + 1$ for any

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 506–517, 2001.

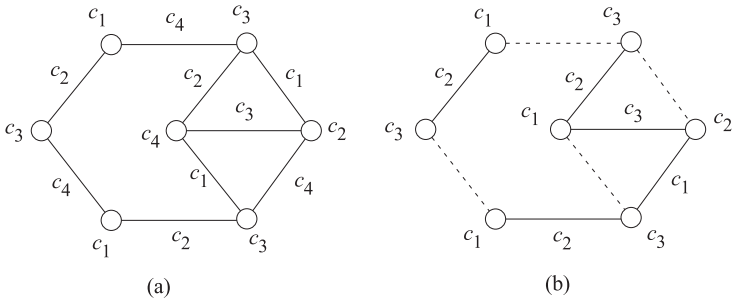


Fig. 1. (a) A total coloring of a graph G with $\chi_t(G) = 4$ colors c_1, c_2, c_3 and c_4, and (b) a semi-total coloring of G for $U = V$ and $F \subseteq E$ with $\chi(G; U, F) = 3$ colors c_1, c_2 and c_3.

s-degenerated graph G [6,8,12]. Let $\chi'(G)$ be the *chromatic index* of G, that is, the minimum number of colors required for an edge-coloring of G. Then clearly $\Delta(G) \leq \chi'(G)$, and hence $\Delta(G)$ is a lower bound on $\chi'(G)$. An s-degenerated graph has a favorable property also on an edge-coloring: $\chi'(G) = \Delta(G)$ if G is an s-degenerated graph and $\Delta(G) \geq 2s$ [13]. Thus there is a simple sufficient condition on $\Delta(G)$, i.e. $\Delta(G) \geq 2s$, for the chromatic index $\chi'(G)$ of an s-degenerated graph G to be equal to the trivial lower bound $\Delta(G)$. However, it has not been known whether there is a simple sufficient condition on $\Delta(G)$ for the total chromatic number $\chi_t(G)$ to be equal to the trivial lower bound $\Delta(G) + 1$.

A graph with bounded tree-width k is called a *partial k-tree*; the formal definition of partial k-trees will be given in Section 2. Any partial k-tree is k-degenerated, but the converse is not always true. Many combinatorial problems can be efficiently solved for partial k-trees with bounded k [1,2,4]. In particular, both the vertex-coloring problem and the edge-coloring problem can be solved in linear time for partial k-trees [2,15]. However, no efficient algorithm has been known for the total coloring problem on partial k-trees. Although the total coloring problem can be solved in polynomial time for partial k-trees by a dynamic programming algorithm, the time complexity $O(n^{1+2^{4(k+1)}})$ is very high [7].

In this paper, we first present a sufficient condition on $\Delta(G)$ for the total chromatic number $\chi_t(G)$ of an s-degenerated graph G to be equal to the trivial lower bound $\Delta(G) + 1$: we prove our main theorem that $\chi_t(G) = \Delta(G) + 1$ if $\Delta(G) \geq 4s + 3$. The condition $\Delta(G) \geq 4s + 3$ for the total chromatic number is simple and interesting, compared to Vizing's condition $\Delta(G) \geq 2s$ for the chromatic index. Our proof immediately yields an efficient algorithm to find a total coloring of G with $\chi_t(G) = \Delta(G) + 1$ colors in time $O(sn^2)$ if $\Delta(G) \geq 4s + 3$, where n is the number of vertices in G. The complexity can be improved to $O(n \log n)$ in particular if $\Delta(G) \geq 6s + 1$ and $s = O(1)$. Hence the total coloring problem can be solved in time $O(n \log n)$ for a fairly large class of graphs including all planar graphs with sufficiently large maximum degree. We

then show that one can find a total coloring of a given partial k-tree G with $\chi_t(G)$ colors in linear time and hence the total coloring problem can be solved in linear time for partial k-trees of bounded k.

2 Preliminaries

We denote by $G = (V, E)$ a simple undirected graph with a vertex set V and an edge set E. Let $n = |V|$ throughout the paper. We denote by $d(v, G)$ the *degree of a vertex v in G*, and by $\Delta(G)$ the *maximum degree of G*. For a set $F \subseteq E$, we denote by $G_F = (V, F)$ the spanning subgraph of G induced by the edge set F. A spanning subgraph of G is called a *forest* of G if each connected component is a tree. A forest of G is called a *linear forest* of G if each connected component is a single isolated vertex or a path. For example, $G_F = (V, F)$ is a linear forest of the graph G in Fig. 1(b) if F consists of the five edges drawn by solid lines.

One of the key ideas in the proof of our main theorem is to introduce a "semi-total coloring," which generalizes vertex-, edge- and total colorings. Let C be a set of colors, let U be a subset of V, and let F be a subset of E. Then a *semi-total coloring of a graph G for U and F* is a mapping $f : U \cup F \to C$ such that

(i) $f(v) \neq f(w)$ if $v, w \in U$ and $(v, w) \in E$;
(ii) $f(e) \neq f(e')$ if $e, e' \in F$ and e and e' share a common end; and
(iii) $f(v) \neq f(e)$ if $v \in U$, $e \in F$ and e is incident to v.

The semi-total coloring is sometimes called a partial total coloring [14]. The vertices in $\overline{U} = V - U$ and the edges in $\overline{F} = E - F$ are not colored by f. Figure 1(b) depicts a semi-total coloring of the graph G in Fig. 1(a) for $U = V$ and F, where $C = \{c_1, c_2, c_3\}$ and all edges in F are drawn by solid lines.

The minimum number of colors required for a semi-total coloring of G for U and F is called the *semi-total chromatic number of G for U and F*, and is denoted by $\chi(G; U, F)$. Then obviously $\Delta(G_F) + 1 \leq \chi(G; V, F)$, where $G_F = (V, F)$ is a spanning subgraph of G. Clearly, a total coloring of G is a semi-total coloring of G for $U = V$ and $F = E$; a vertex-coloring of G is a semi-total coloring of G for $U = V$ and $F = \emptyset$; and an edge-coloring of G is a semi-total coloring of G for $U = \emptyset$ and $F = E$.

Another idea is a "superimposing" of colorings. Suppose that g is a semi-total coloring of a graph $G = (V, E)$ for $U = V$ and $F \subseteq E$, h is an edge-coloring of $G_{\overline{F}} = (V, E - F)$, and g and h use no common color. Then, superimposing g on h, one can obtain a total coloring f of G, and hence

$$\chi_t(G) \leq \chi(G; V, F) + \chi'(G_{\overline{F}}). \tag{1}$$

Unfortunately, the total coloring f constructed from g and h may use more than $\chi_t(G)$ colors even if g uses the minimum number $\chi(G; V, F)$ of colors and h uses the minimum number $\chi'(G_{\overline{F}})$ of colors, because the equality in Eq. (1) does not always hold. For example, for the graph G in Fig. 1(b), $\chi_t(G) = 4$, $\chi(G; V, F) = 3$, $\chi'(G_{\overline{F}}) = 2$, and hence $\chi_t(G) < \chi(G; V, F) + \chi'(G_{\overline{F}})$. However,

we will show in Section 3 as the main theorem that if G is an s-degenerated graph and $\Delta(G) \geq 4s+3$ then there is a subset $F \subset E$ such that the equality in Eq. (1) holds and $\chi(G;V,F) + \chi'(G_{\overline{F}}) = \Delta(G)+1$, that is,

$$\chi_{\rm t}(G) = \chi(G;V,F) + \chi'(G_{\overline{F}}) = \Delta(G)+1. \tag{2}$$

We will show in Section 4 that one can efficiently find a semi-total coloring g of G for V and F with $\chi(G;V,F)$ colors and an edge-coloring h of $G_{\overline{F}}$ with $\chi'(G_{\overline{F}})$ colors, and hence one can efficiently find a total coloring f of G with $\chi_{\rm t}(G)$ colors simply by superimposing g and h.

We now recursively define a k-tree: a graph $G=(V,E)$ is a *k-tree* if it is a complete graph of k vertices or it has a vertex $v \in V$ of degree k whose neighbors induce a clique of size k and the graph $G-\{v\}$ obtained from G by deleting the vertex v and all edges incident to v is again a k-tree. We then define a partial k-tree: a graph is a *partial k-tree* if it is a subgraph of a k-tree [1,2,15]. The graph in Fig. 1(a) is indeed a partial 3-tree. In this paper we assume that $k = O(1)$.

For an integer $s \geq 1$, a graph G is defined to be *s-degenerated* (or *s-inductive*) if G can be reduced to a trivial graph by the successive removal of vertices having degree at most s [6,8,16]. We do not assume that $s = O(1)$. By the definition of an s-degenerated graph $G=(V,E)$, there exists a numbering $\varphi : V \to \{1,2,\cdots,n\}$ such that any vertex $v \in V$ has at most s neighbors numbered by φ with integers larger than $\varphi(v)$, that is,

$$|\{x \in V : (v,x) \in E,\ \varphi(v) < \varphi(x)\}| \leq s.$$

Such a numbering φ is called an *s-numbering of G*. An s-numbering of any s-degenerated graph G can be found in linear time [9].

For a vertex v in a graph $G=(V,E)$ and a numbering $\varphi : V \to \{1,2,\cdots,n\}$, we write

$$\begin{aligned} E^{\rm fw}_{\varphi}(v,G) &= \{(v,x) \in E : \varphi(v) < \varphi(x)\};\\ E^{\rm bw}_{\varphi}(v,G) &= \{(v,x) \in E : \varphi(v) > \varphi(x)\};\\ d^{\rm fw}_{\varphi}(v,G) &= |E^{\rm fw}_{\varphi}(v,G)|;\ \text{and}\\ d^{\rm bw}_{\varphi}(v,G) &= |E^{\rm bw}_{\varphi}(v,G)|. \end{aligned}$$

The edges in $E^{\rm fw}_{\varphi}(v,G)$ are called the *forward edges of v*, and those in $E^{\rm bw}_{\varphi}(v,G)$ the *backward edges of v*. Clearly $d(v,G) = d^{\rm fw}_{\varphi}(v,G) + d^{\rm bw}_{\varphi}(v,G)$. A graph G is s-degenerated if and only if there is a numbering φ such that $d^{\rm fw}_{\varphi}(v,G) \leq s$ for any vertex $v \in V$. An s-degenerated graph has the following two favorable properties on colorings, which have been mentioned in Introduction.

Lemma 1. *For any s-degenerated graph G, the following* (a) *and* (b) *hold*:

(a) $\chi(G) \leq s+1$ [6,8,9,12]; *and*
(b) *if* $\Delta(G) \geq 2s$, *then* $\chi'(G) = \Delta(G)$ [13,16].

3 Main Theorem

In this section we prove the following main theorem.

Theorem 1. *If G is an s-degenerated graph and $\Delta(G) \geq 4s+3$, then $\chi_{\rm t}(G) = \Delta(G)+1$.*

A result by Borodin *et al.* on a "total *list* coloring" [3, Theorem 7] implies that $\chi_{\rm t}(G) = \Delta(G)+1$ if G is an s-degenerated graph and $\Delta(G) \geq 2s^2$. Theorem 1 is better than these results.

We show in the remainder of this section that there is a subset $F \subset E$ satisfying Eq. (2). F will be found as a union of $s+1$ edge-disjoint linear forests of G.

We first show in the following lemma that G can be decomposed to $s+1$ edge-disjoint forests.

Lemma 2. *If $G=(V,E)$ is an s-degenerated graph, then there exists a partition $\{F_1, F_2, \cdots, F_{s+1}\}$ of E such that, for any index $j \in \{1,2,\cdots,s+1\}$,*

(a) *G_{F_j} is a forest; and*
(b) *$d(v, G_{F_j}) \geq 3$ if $d(v,G) \geq 4s+3$.*

Proof. Let $G=(V,E)$ be an s-degenerated graph, and let $\varphi : V \to \{1,2,\cdots,n\}$ be an s-numbering of G.

We first find $F_1, F_2, \cdots, F_{s+1}$. Construct a new graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ from G as follows: for any vertex $v \in V$,

- let $t = \lceil d^{\rm bw}_\varphi(v,G)/(s+1) \rceil$, and replace v with its $t+1$ copies $v_{\rm fw}$, $v^1_{\rm bw}, v^2_{\rm bw}$, $\cdots, v^t_{\rm bw}$;
- attach all forward edges in $E^{\rm fw}_\varphi(v,G)$ to the copy $v_{\rm fw}$; and
- let $\{E^1_{\rm bw}, E^2_{\rm bw}, \cdots, E^t_{\rm bw}\}$ be any partition of the set $E^{\rm bw}_\varphi(v,G)$ of backward edges such that

$$|E^i_{\rm bw}| \begin{cases} = s+1 \text{ if } 1 \leq i \leq t-1; \\ \leq s+1 \text{ if } i = t, \end{cases} \tag{3}$$

  and attach all edges in $E^i_{\rm bw}$ to the copy $v^i_{\rm bw}$ for each $i = 1,2,\cdots,t$.

Clearly, $\widetilde{G}$ is bipartite. Since φ is an s-numbering of G, $d(v_{\rm fw}, \widetilde{G}) = d^{\rm fw}_\varphi(v,G) \leq s$ for any vertex $v \in V$. By Eq. (3) $d(v^i_{\rm bw}, \widetilde{G}) \leq s+1$ for any vertex $v \in V$ and any index $i \in \{1,2,\cdots,t\}$. Thus we have $\Delta(\widetilde{G}) \leq s+1$.

Since $\widetilde{G}$ is bipartite and $\Delta(\widetilde{G}) \leq s+1$, König's theorem implies that $\chi'(\widetilde{G}) = \Delta(\widetilde{G}) \leq s+1$ [6,8], and hence $\widetilde{G}$ has an edge-coloring $f : \widetilde{E} \to C$ for a set $C = \{c_1, c_2, \cdots, c_{s+1}\}$ of $s+1$ colors. For each color $c_j \in C$, let $\widetilde{F}_j$ be the color class of c_j, that is, $\widetilde{F}_j = \{e \in \widetilde{E} : f(e) = c_j\}$, and let F_j be the set of edges in E corresponding to $\widetilde{F}_j$. Since f is an edge-coloring of $\widetilde{G}$, $\{\widetilde{F}_1, \widetilde{F}_2, \cdots, \widetilde{F}_{s+1}\}$ is a partition of $\widetilde{E}$, and hence $\{F_1, F_2, \cdots, F_{s+1}\}$ is a partition of E. Thus we have found $F_1, F_2, \cdots, F_{s+1}$.

We then prove that F_j found as above satisfies (a) and (b) for any index $j \in \{1, 2, \cdots, s+1\}$.

(a) It suffices to prove that the s-numbering φ of G is indeed a 1-numbering of G_{F_j}, that is, $d^{\mathrm{fw}}_{\varphi}(v, G_{F_j}) \leq 1$ for any vertex $v \in V$. By the construction of $\widetilde{G}$, all forward edges of v in G are attached to the copy v_{fw} in $\widetilde{G}$. At most one of them is colored with c_j by f since f is an edge-coloring of $\widetilde{G}$. Thus $\widetilde{F}_j$ contains at most one edge incident to v_{fw}, and hence we have $d^{\mathrm{fw}}_{\varphi}(v, G_{F_j}) \leq 1$.

(b) Let v be any vertex in V with $d(v, G) \geq 4s + 3$.

We first claim that $d(v^i_{\mathrm{bw}}, \widetilde{G}) = s+1$ for each $i \in \{1, 2, 3\}$. By the construction of $\widetilde{G}$, $d(v^i_{\mathrm{bw}}, \widetilde{G}) = s + 1$ if $i \leq \left\lfloor d^{\mathrm{bw}}_{\varphi}(v, G)/(s+1) \right\rfloor$. It therefore suffices to prove that $\left\lfloor d^{\mathrm{bw}}_{\varphi}(v, G)/(s+1) \right\rfloor \geq 3$. Clearly, $d^{\mathrm{fw}}_{\varphi}(v, G) \leq s$ and $d(v, G) = d^{\mathrm{fw}}_{\varphi}(v, G) + d^{\mathrm{bw}}_{\varphi}(v, G)$. Hence we have

$$\left\lfloor \frac{d^{\mathrm{bw}}_{\varphi}(v, G)}{s+1} \right\rfloor = \left\lfloor \frac{d(v, G) - d^{\mathrm{fw}}_{\varphi}(v, G)}{s+1} \right\rfloor \geq \left\lfloor \frac{4s + 3 - s}{s+1} \right\rfloor = 3. \tag{4}$$

Thus we have proved the claim.

We then prove that $d(v, G_{F_j}) \geq 3$. The edge-coloring f of $\widetilde{G}$ uses exactly $s+1$ colors in C, and $d(v^i_{\mathrm{bw}}, \widetilde{G}) = s + 1$ for each $i \in \{1, 2, 3\}$. Therefore exactly one of the edges incident to v^i_{bw} in $\widetilde{G}$ is colored with $c_j \in C$ by f. We thus have $d(v, G_{F_j}) \geq 3$. □

One can construct a linear forest from a forest as in the following lemma.

Lemma 3. *Let $T = (V, F)$ be a forest, let $S = \{v \in V : d(v, T) \geq 3\}$, and let U be any subset of S. Then T has a linear forest $T_L = (V, L)$, $L \subseteq F$, such that every vertex in U is an end of a path in T_L, and every vertex in $S - U$ is an interior vertex of a path in T_L, that is,*

$$d(v, T_L) = \begin{cases} 1 \text{ if } v \in U; \text{ and} \\ 2 \text{ if } v \in S - U. \end{cases} \tag{5}$$

Furthermore L can be found in linear time.

Sketchy Proof. Regard each tree in forest T as a rooted tree. Then one can easily find a linear forest of the tree by the breadth-first search. □

By Lemmas 2 and 3 one can find $s+1$ linear forests $G_{L_1}, G_{L_2}, \cdots, G_{L_{s+1}}$ of G as in the following lemma.

Lemma 4. *If $G = (V, E)$ is an s-degenerated graph and $\Delta(G) \geq 4s + 3$, then for any partition $\{U_1, U_2, \cdots, U_{s+1}\}$ of V there exist mutually disjoint subsets $L_1, L_2, \cdots, L_{s+1}$ of E such that*

(a) *for each $j \in \{1, 2, \cdots, s+1\}$, G_{L_j} is a linear forest, and*

$$d(v, G_{L_j}) \leq \begin{cases} 1 \text{ if } v \in U_j; \\ 2 \text{ if } v \in V - U_j; \end{cases} \tag{6}$$

(b) $\Delta(G_F) = 2s+1$, *where* $F = L_1 \cup L_2 \cup \cdots \cup L_{s+1}$; *and*
(c) $\Delta(G_F) + \Delta(G_{\overline{F}}) = \Delta(G)$, *where* $\overline{F} = E - F$.

Proof. Let $G = (V, E)$ be an s-degenerated graph, and let $\Delta(G) \geq 4s+3$. We find $L_1, L_2, \cdots, L_{s+1}$ as follows.

We first construct a new graph $G^* = (V^*, E^*)$ from G as follows: for each vertex $v \in V$ with $d(v, G) < 4s+3$, add $(4s+3) - d(v, G)$ dummy vertices and join each of them with v by a dummy edge. Clearly

$$d(v, G^*) = \begin{cases} d(v, G) & \text{if } v \in V \text{ and } d(v, G) \geq 4s+3; \\ 4s+3 & \text{if } v \in V \text{ and } d(v, G) < 4s+3; \text{ and} \\ 1 & \text{if } v \in V^* - V, \end{cases} \tag{7}$$

and hence

$$V = \{v \in V^* : d(v, G^*) \geq 4s+3\}. \tag{8}$$

Since $\Delta(G) \geq 4s+3$,

$$\Delta(G^*) = \Delta(G). \tag{9}$$

We then find $s+1$ forests of G^*. Since G is s-degenerated, G^* is also s-degenerated. Therefore, applying Lemma 2 to G^*, one can know that there exists a partition $\{F_1, F_2, \cdots, F_{s+1}\}$ of E^* such that, for any index $j \in \{1, 2, \cdots, s+1\}$,

(i) $G^*_{F_j}$ is a forest; and
(ii) $d(v, G^*_{F_j}) \geq 3$ if $d(v, G^*) \geq 4s+3$.

By (ii) and Eqs. (7) and (8) we have $V = \{v \in V^* : d(v, G^*_{F_j}) \geq 3\}$.

We then find $s+1$ linear forests of G^*. Let $\{U_1, U_2, \cdots, U_{s+1}\}$ be any partition of V. For each $j \in \{1, 2, \cdots, s+1\}$, apply Lemma 3 to $T = G^*_{F_j}$, $S = V = \{v \in V^* : d(v, G^*_{F_j}) \geq 3\}$ and $U = U_j \subseteq V$, then one can know that the forest $G^*_{F_j}$ has a linear forest $G^*_{L^*_j} = (V^*, L^*_j)$ such that

$$d(v, G^*_{L^*_j}) = \begin{cases} 1 \text{ if } v \in U_j; \text{ and} \\ 2 \text{ if } v \in V - U_j. \end{cases} \tag{10}$$

Since $L^*_j \subseteq F_j$, $1 \leq j \leq s+1$, and $F_1, F_2, \cdots, F_{s+1}$ are mutually disjoint with each other, $L^*_1, L^*_2, \cdots, L^*_{s+1}$ are also mutually disjoint with each other.

We then find $L_1, L_2, \cdots, L_{s+1}$ from $L^*_1, L^*_2, \cdots, L^*_{s+1}$; for each $j \in \{1, 2, \cdots, s+1\}$, let L_j be the set of all non-dummy edges in L^*_j, that is, $L_j = L^*_j \cap E$. Then $L_j \subseteq L^*_j$. Furthermore, one can easily observe that

$$d(v, G_{L_j}) \begin{cases} \leq d(v, G^*_{L^*_j}) \text{ if } v \in V; \\ = d(v, G^*_{L^*_j}) \text{ if } v \in V \text{ and } d(v, G) \geq 4s+3. \end{cases} \tag{11}$$

Since $L^*_1, L^*_2, \cdots, L^*_{s+1}$ are mutually disjoint with each other, $L_1, L_2, \cdots, L_{s+1}$ are also mutually disjoint with each other. Thus we have found $L_1, L_2, \cdots, L_{s+1}$.

We shall prove that $L_1, L_2, \cdots, L_{s+1}$ and $F = L_1 \cup L_2 \cup \cdots \cup L_{s+1}$ satisfy (a)–(c).

(a) Since $L_j \subseteq L_j^*$ and $G^*_{L_j^*}$ is a linear forest of G^*, G_{L_j} is a linear forest of G. Let v be any vertex in V. If $v \in U_j$, then by Eqs. (10) and (11) we have $d(v, G_{L_j}) \leq d(v, G^*_{L_j^*}) = 1$. Similarly, if $v \in V - U_j$, then we have $d(v, G_{L_j}) \leq d(v, G^*_{L_j^*}) = 2$.

(b) We first prove that $\Delta(G_F) \leq 2s+1$. Let $F^* = L_1^* \cup L_2^* \cup \cdots \cup L_{s+1}^*$. Let v be any vertex in V, and let j be the index such that $v \in U_j$. Then by Eq. (10) we have

$$d(v, G^*_{F^*}) = 2s+1. \tag{12}$$

Since $F \subseteq F^*$, we have $d(v, G_F) \leq d(v, G^*_{F^*}) = 2s+1$. Thus we have $\Delta(G_F) \leq 2s+1$.

We then prove that $\Delta(G_F) \geq 2s+1$. Since $\Delta(G) \geq 4s+3$, G has a vertex v with $d(v, G) \geq 4s+3$. Since $F = L_1 \cup L_2 \cup \cdots \cup L_{s+1}$, we have

$$d(v, G_F) = \sum_{i=1}^{s+1} d(v, G_{L_i}). \tag{13}$$

By Eqs. (11), (12) and (13) we have $d(v, G_F) = d(v, G^*_{F^*}) = 2s+1$. We thus have $2s+1 \leq \Delta(G_F)$.

(c) Clearly, $\Delta(G_F) + \Delta(G_{\overline{F}}) \geq \Delta(G)$ for any set $F \subseteq E$. We shall therefore prove that $\Delta(G_F) + \Delta(G_{\overline{F}}) \leq \Delta(G)$, that is, $\Delta(G_{\overline{F}}) \leq \Delta(G) - \Delta(G_F)$. Since $E \subseteq E^*$ and $F = F^* \cap E$,

$$E - F \subseteq E^* - F^*. \tag{14}$$

For any vertex $v \in V$, by (b) above and Eqs. (9), (12) and (14) we have

$$\begin{aligned} d(v, G_{\overline{F}}) &= d(v, G_{E-F}) \\ &\leq d(v, G^*_{E^*-F^*}) \\ &= d(v, G^*) - d(v, G^*_{F^*}) \\ &\leq \Delta(G^*) - (2s+1) \\ &= \Delta(G) - \Delta(G_F). \end{aligned}$$

Thus we have $\Delta(G_{\overline{F}}) \leq \Delta(G) - \Delta(G_F)$. □

By Lemma 1(a) any s-degenerated graph G has a vertex-coloring with $s+1$ colors. Choose the set of color classes as the partition $\{U_1, U_2, \cdots, U_{s+1}\}$ in Lemma 4. Then there is a subset $F \subset E$ satisfying Eq. (2), as shown in the following theorem.

Theorem 2. *If $G = (V, E)$ is an s-degenerated graph and $\Delta(G) \geq 4s+3$, then there exists a subset F of E such that*

(a) $\chi(G; V, F) = \Delta(G_F) + 1$;
(b) $\chi'(G_{\overline{F}}) = \Delta(G_{\overline{F}})$, *where* $\overline{F} = E - F$;
(c) $\Delta(G_F) + \Delta(G_{\overline{F}}) = \Delta(G)$; *and*
(d) $\chi_t(G) = \chi(G; V, F) + \chi'(G_{\overline{F}}) = \Delta(G) + 1$.

Proof. Let $G = (V, E)$ be an s-degenerated graph, and let $\Delta(G) \geq 4s+3$. Since G is s-degenerated, by Lemma 1(a) $\chi(G) \leq s+1$ and hence G has a vertex-coloring $f : V \to C$ for a set $C = \{c_1, c_2, \cdots, c_{s+1}\}$ of $s+1$ colors. For each color $c_j \in C$, let U_j be the color class of c_j, that is, $U_j = \{v \in V : f(v) = c_j\}$. Then $\{U_1, U_2, \cdots, U_{s+1}\}$ is a partition of V and each of $U_1, U_2, \cdots, U_{s+1}$ is an independent set of G. By Lemma 4 for the partition $\{U_1, U_2, \cdots, U_{s+1}\}$ there exist mutually disjoint subsets $L_1, L_2, \cdots, L_{s+1}$ of E satisfying the conditions (a)–(c) in Lemma 4. Since the condition (c) in Theorem 2 is the same as (c) in Lemma 4, we shall show that $F = L_1 \cup L_2 \cup \cdots \cup L_{s+1}$ satisfies the conditions (a), (b) and (d) in Theorem 2.

(a) Clearly, $\chi(G; V, F) \geq \Delta(G_F) + 1$ for any set $F \subseteq E$. Therefore it suffices to prove that $\chi(G; V, F) \leq \Delta(G_F) + 1$.

For each $j \in \{1, 2, \cdots, s+1\}$, we first construct a semi-total coloring g_j of G for U_j and L_j with two colors, as follows. Since G_{L_j} is a linear forest by Lemma 4(a), we have $\chi'(G_{L_j}) = \Delta(G_{L_j}) \leq 2$ and hence G_{L_j} has an edge-coloring $f_j : L_j \to C_j$ for a set C_j of two colors. Since $|C_j| = 2$ and $d(v, G_{L_j}) \leq 1$ for each vertex $v \in U_j$ by Eq. (6), there is a color $c_v \in C_j$ such that the edge-coloring f_j does not assign c_v to any edge incindet to v. We then extend the mapping $f_j : L_j \to C_j$ to a mapping $g_j : U_j \cup L_j \to C_j$, as follows:

$$g_j(x) = \begin{cases} f_j(x) & \text{for each edge } x \in L_j; \text{ and} \\ c_x & \text{for each vertex } x \in U_j. \end{cases} \tag{15}$$

Then g_j is a semi-total coloring of G for U_j and L_j, that is, g_j satisfies the three conditions (i), (ii) and (iii) on a semi-total coloring mentioned in Section 2, as follows. Since U_j is an independent set of G, clearly g_j satisfies the condition (i) for $U = U_j$ and $F = L_j$. Since f_j is an edge-coloring of G_{L_j}, g_j satisfies the condition (ii). By Eq. (15) g_j satisfies the condition (iii).

From $g_1, g_2, \cdots, g_{s+1}$ above we then construct a semi-total coloring g of G for V and F with $\Delta(G_F) + 1$ colors, as follows. One may assume that any two of the semi-total colorings $g_1, g_2, \cdots, g_{s+1}$ use no common color, that is, $C_p \cap C_q = \emptyset$ for any p and q, $1 \leq p < q \leq s+1$. Superimpose $g_1, g_2, \cdots, g_{s+1}$, and let $g : V \cup F \to C_1 \cup C_2 \cup \cdots \cup C_{s+1}$ be the resulting coloring. Then one can easily observe that g is a semi-total coloring of G for V and F. The semi-total coloring g uses $\sum_{i=1}^{s+1} |C_i| = 2(s+1)$ colors, and $\Delta(G_F) = 2s+1$ by Lemma 4(b). Therefore we have $\chi(G; V, F) \leq 2(s+1) = \Delta(G_F) + 1$.

(b) Since G is s-degenerated, the subgraph $G_{\overline{F}}$ of G is also s-degenerated. Since $\Delta(G) \geq 4s+3$, by the conditions (b) and (c) in Lemma 4 we have $\Delta(G_{\overline{F}}) = \Delta(G) - \Delta(G_F) \geq (4s+3) - (2s+1) = 2s+2 > 2s$. Therefore, by Lemma 1(b) we have $\chi'(G_{\overline{F}}) = \Delta(G_{\overline{F}})$.

(d) By (a), (b) and (c) we have $\chi(G; V, F) + \chi'(G_{\overline{F}}) = \Delta(G_F) + 1 + \Delta(G_{\overline{F}}) = \Delta(G) + 1$. Thus we shall prove that $\chi_{\mathrm{t}}(G) = \chi(G; V, F) + \chi'(G_{\overline{F}})$.

By Eq. (1) $\chi_{\mathrm{t}}(G) \leq \chi(G; V, F) + \chi'(G_{\overline{F}})$. Clearly $\chi_{\mathrm{t}}(G) \geq \Delta(G) + 1 = \chi(G; V, F) + \chi'(G_{\overline{F}})$. Thus we have $\chi_{\mathrm{t}}(G) = \chi(G; V, F) + \chi'(G_{\overline{F}}) = \Delta(G) + 1$. □

Theorem 2(d) implies Theorem 1.

4 Algorithms

From the proofs of Lemmas 2–4 and Theorem 2, one can know that the following algorithm correctly finds a total coloring of an s-degenerated graph $G = (V, E)$ with $\Delta(G) + 1$ colors if $\Delta(G) \geq 4s + 3$.

[**Total-Coloring Algorithm**]

Step 1. Find a vertex-coloring of a given s-degenerated graph G with $s + 1$ colors, and let $\{U_1, U_2, \cdots, U_{s+1}\}$ be the set of color classes.

Step 2. Construct a graph $G^* = (V^*, E^*)$ from G by adding dummy vertices and edges, as in the proof of Lemma 4.

Step 3. Construct a bipartite graph $\widetilde{G^*} = (\widetilde{V^*}, \widetilde{E^*})$ from G^* by splitting each vertex in G^*, as in the proof of Lemma 2. Note that $\Delta(\widetilde{G^*}) \leq s + 1$.

Step 4. Find an edge-coloring of the bipartite graph $\widetilde{G^*}$ with $s + 1$ colors, and let $\{\widetilde{F_1^*}, \widetilde{F_2^*}, \cdots, \widetilde{F_{s+1}^*}\}$ be the set of color classes. Let $\{F_1, F_2, \cdots, F_{s+1}\}$ be the partition of E^* corresponding to $\{\widetilde{F_1^*}, \widetilde{F_2^*}, \cdots, \widetilde{F_{s+1}^*}\}$, where $G_{F_j}^*$, $1 \leq j \leq s + 1$, is a forest of G^*.

Step 5. From each forest $G_{F_j}^*$ of G^*, $1 \leq j \leq s + 1$, find a linear forest $G_{L_j^*}^*$ of G^* such that

$$d(v, G_{L_j^*}^*) = \begin{cases} 1 \text{ if } v \in U_j; \\ 2 \text{ if } v \in V - U_j, \end{cases}$$

as in the proof of Lemma 3, where U_j is a color class found in Step 1.

Step 6. From each linear forest $G_{L_j^*}^*$ of G^*, obtain a linear forest G_{L_j} of G such that

$$d(v, G_{L_j}) \leq \begin{cases} 1 \text{ if } v \in U_j; \\ 2 \text{ if } v \in V - U_j, \end{cases}$$

by deleting all dummy vertices and edges as in the proof of Lemma 4.

Step 7. For each j, find an edge-coloring f_j of the linear forest G_{L_j} with two colors, and extend f_j to a semi-total coloring g_j of G for U_j and L_j as in the proof of Theorem 2.

Step 8. Superimposing $g_1, g_2, \cdots, g_{s+1}$, obtain a semi-total coloring g of G for V and $F = L_1 \cup L_2 \cup \cdots \cup L_{s+1}$ with $\Delta(G_F) + 1$ colors, as in the proof of Theorem 2.

Step 9. Find an edge-coloring h of $G_{\overline{F}}$ with $\Delta(G_{\overline{F}})$ colors.

Step 10. Superimposing g and h, obtain a total coloring of G with $\Delta(G) + 1$ colors.

We then show that all steps can be done in time $O(sn^2)$, using an algorithm for edge-coloring bipartite graphs [5] and an algorithm for edge-coloring s-degenerated graphs [15,16].

One can easily find the vertex-coloring of G in time $O(sn)$ by a simple greedy algorithm based on an s-numbering of G [6,8,9,12]. Note that G has at most sn edges. Thus Step 1 can be done in time $O(sn)$.

By the construction of the graph G^*, we have $|V^* - V| = |E^* - E| \leq (4s+3)n$ and hence

$$|V^*| \leq n + (4s+3)n = 4(s+1)n, \text{ and}$$
$$|E^*| \leq sn + (4s+3)n = (5s+3)n.$$

Thus one can construct the graph G^* in time $O(sn)$, and hence Step 2 can be done in time $O(sn)$.

Clearly

$$|\widetilde{E^*}| = |E^*| \leq (5s+3)n, \text{ and}$$
$$|\widetilde{V^*}| \leq 2|\widetilde{E^*}| \leq 2(5s+3)n.$$

Therefore one can construct $\widetilde{G^*}$ from G^* in time $O(sn)$. Thus Step 3 can be done in time $O(sn)$.

Since $\widetilde{G^*}$ is bipartite and $\Delta(\widetilde{G^*}) \leq s+1$, one can find the edge-coloring of $\widetilde{G^*}$ in time $O(|\widetilde{E^*}| \log \Delta(\widetilde{G^*}) = O(sn \log s)$ [5]. Note that $s \leq n$. Thus Step 4 can be done in time $O(sn \log n)$.

By Lemma 2, for each forest $G^*_{F_j}$, $1 \leq j \leq s+1$, one can find the linear forest $G^*_{L^*_j}$ in time $O(|V^*|) = O(sn)$. Therefore the $s+1$ linear forests $G^*_{L^*_1}, G^*_{L^*_2}, \cdots, G^*_{L^*_{s+1}}$ can be found in time $O((s+1)sn) = O(s^2n)$. Thus Step 5 can be done in time $O(s^2n)$.

From each linear forest $G^*_{L^*_j}$, $1 \leq j \leq s+1$, one can obtain the linear forest G_{L_j} in time $O(|L^*_j|) = O(sn)$ simply by deleting dummy vertices and edges. Therefore one can obtain the $s+1$ linear forests $G_{L_1}, G_{L_2}, \cdots, G_{L_{s+1}}$ in time $O((s+1)sn) = O(s^2n)$. Thus Step 6 can be done in time $O(s^2n)$.

For each j, $1 \leq j \leq s+1$, one can easily find an edge-coloring f_j of the linear forest G_{L_j} with two colors in time $O(|L_j|) = O(n)$, and can extend f_j to the semi-total coloring g_j in time $O(n)$. Therefore Step 7 can be done in time $O(sn)$.

Superimposing $g_1, g_2, \cdots, g_{s+1}$, one can obtain the semi-total coloring g of G for V and F in time $O(sn)$. Thus Step 8 can be done in time $O(sn)$.

Since $G_{\overline{F}}$ is s-degenerated, one can find the edge-coloring h of $G_{\overline{F}}$ in time $O(sn^2)$ [10, 15 p.604, 16 p.8]. Therefore Step 9 can be done in time $O(sn^2)$.

Superimposing g and h, one can obtain a total coloring of G with $\chi_t(G) = \Delta(G) + 1$ colors in time $O(sn)$. Thus Step 10 can be done in time $O(sn)$.

Thus all Steps 1–10 above can be done in time $O(sn^2)$, and hence we have the following theorem.

Theorem 3. *A total coloring of an s-degenerated graph G using $\chi_t(G) = \Delta(G) + 1$ colors can be found in time $O(sn^2)$ if $\Delta(G) \geq 4s+3$.*

The complexity $O(sn^2)$ can be improved as in the following two theorems.

Theorem 4. *A total coloring of an s-degenerated graph G using $\chi_t(G) = \Delta(G) + 1$ colors can be found in time $O(n \log n)$ if $\Delta(G) \geq 6s+1$ and $s = O(1)$.*

Sketchy Proof. Use an $O(n \log n)$ algorithm in [16] to find an edge-coloring of $G_{\overline{F}}$. □

Theorem 5. *The total coloring problem can be solved in linear time for partial k-trees G with bounded k.*

Sketchy Proof. For the case where $\Delta(G) < 4k + 3$, use the algorithm in [7] to find a total coloring of G in time $O(n\chi_{\mathrm{t}}^{2^{4(k+1)}}) = O(n)$. For the case where $\Delta(G) \geq 4k + 3$, use our algorithm to find a total coloring of G, but use a linear-time algorithm in [15] to find an edge-coloring of $G_{\overline{F}}$ in Step 9. □

References

1. S. Arnborg and J. Lagergren, Easy problems for tree-decomposable graphs, J. Algorithms, 12(2), pp. 308–340, 1991.
2. H. L. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees, J. Algorithms, 11(4), pp. 631–643, 1990.
3. O. V. Borodin, A. V. Kostochka and D. R. Woodall, List edge and list total colourings of multigraphs, J. Combinatorial Theory, Series B, 71, pp. 184–204, 1997.
4. R. B. Borie, R. G. Parker and C. A. Tovey, Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families, Algorithmica, 7, pp. 555–581, 1992.
5. R. Cole, K. Ost and S. Schirra, Edge-coloring bipartite multigraphs in $O(E \log D)$ time, Combinatorica, 21, pp. 5–12, 2001.
6. R. Diestel, Graph Theory, Springer, New York, 1997.
7. S. Isobe, X. Zhou and T. Nishizeki, A polynomial-time algorithm for finding total colorings of partial k-trees, Int. J. Found. Comput. Sci., 10(2), pp. 171–194, 1999.
8. T. R. Jensen and B. Toft, Graph Coloring Problems, John Wiley & Sons, New York, 1995.
9. D. Matula and L. Beck, Smallest-last ordering and clustering and graph coloring algorithms, J. Assoc. Comput. Mach., 30, pp. 417–427, 1983.
10. T. Nishizeki and N. Chiba, Planar Graphs: Theory and Algorithms, North-Holland, Amsterdam, 1988.
11. A. Sánchez-Arroyo. Determining the total colouring number is NP-hard, Discrete Math., 78, pp. 315–319, 1989.
12. G. Szekeres and H. Wilf, An inequality for the chromatic number of a graph, J. Combinatorial Theory, 4, pp. 1–3, 1968.
13. V. G. Vizing, Critical graphs with given chromatic class (in Russian), Metody Discret Analiz., 5, pp. 9–17, 1965.
14. H. P. Yap, Total Colourings of Graphs, Lect. Notes in Math., 1623, Springer, Berlin, 1996.
15. X. Zhou, S. Nakano and T. Nishizeki, Edge-coloring partial k-trees, J. Algorithms, 21, pp. 598–617, 1996.
16. X. Zhou and T. Nishizeki, Edge-coloring and f-coloring for various classes of graphs, J. Graph Algorithms and Applications, `http://www.cs.brown.edu/publications/jgaa/`, 3(1), pp. 1–18, 1999.

Decidable Properties of Graphs of All-Optical Networks

Luciano Margara[1] and Janos Simon[2]

[1] Computer Science Department, University of Bologna.
margara@cs.unibo.it
[2] Computer Science Department, University of Chicago.
simon@cs.uchicago.edu

Abstract. We examine several decidability questions suggested by questions about all-optical networks, related to the *gap* between maximal load and number of colors (wavelengths) needed for a legal routing on a fixed graph. We prove the *multiple fiber conjecture*: for every fixed graph G there is a number L_G such that in the communication network with L_G parallel fibers for each edge of G, there is no gap (for any load). We prove that for a fixed graph G the existence of a gap is computable, and give an algorithm to compute it. We develop a decomposition theory for paths, defining the notion of *prime* sets of paths that are finite building blocks for all loads on a fixed graph. Properties of such decompositions yield our theorems.

1 Introduction

The problem: standard model. Variants of the *wavelength assignment problem* for all-optical networks have been extensively studied both because of possible applications, and because of the intrinsic interest of the combinatorial optimization problems suggested by the model [2,3,4,5,7,9,10,11,12,15,16,17,19,23]. We first present the main results informally: a more precise formulation, and justification for the model will follow.

Consider a fixed (possibly directed) multigraph G (the *interconnection graph*) and a collection I of paths (associated to point-to-point communication requests), find a (minimum cardinality) coloring of the paths so that any two paths sharing an edge have different colors. In this paper we do not consider the problem of how to associate paths to communication requests specified by pairs of source/destination nodes. We will assume that the routing scheme is given as part of the input, or that it is uniquely determined by the topology of the network we are considering (as in the case of optical trees). The problem of assigning colors (wavelengths) to a given set of communication paths, so that paths sharing an edge get different colors is the wavelength assignment problem.

The problem of finding an optimal color assignment is NP-hard even if we restrict our attention to very simple 1-fiber network families such as *rings* or *trees* [4,5,6]. In some cases, there exist polynomial time greedy algorithms (see for example [21,9]) which provide approximately optimal solutions for this problem.

A natural lower bound to the number of colors necessary is the *maximum load* $L(I)$, the maximum number of paths through an edge. The maximum load bound

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 518–529, 2001.

is not tight (consider the clockwise oriented triangle and the instance consisting of the requests {[1,3], [2,1], [3,2]}. The maximum load per edge is 2, and 3 colors are necessary). We call the (worst) ratio of the number of colors to $L(I)$ the *gap* of the graph (this is a characteristic of the graph). Precise definitions are given at the end of the section.

It would be very useful if we knew that for a given graph number of colors equal to the maximum load was achievable, or if we had a precise estimate of how bad this bound was. If an approximation algorithm achieved a number of colors close to $(gap)\times(L(I))$, we could accept the output as reasonably close to the hard to compute true optimum.

Multiple fiber model. The multiple fiber model has been introduced independently in [15] and in [12]. In this new model each edge of the interconnection graph consists of k parallel fibers and the same wavelength can be used by at most k distinct paths traveling through the same edge.

Evidence that in the multiple fiber model the gap between number of wavelengths and load (now the load is the load in the 1-fiber model divided by k) can be significantly reduced (at least for some specific network topologies) is given in [15] and in [12]. For example, in the clockwise oriented triangle mentioned above, 2 fibers suffice to eliminate the gap.

In [15] we have stated the following **Multiple Fiber Conjecture**: For any fixed graph G, there is a k (independent of the requests), so that if we consider the graph with k parallel edges at each edge of G, there is no asymptotic gap.

Clearly, for any fixed instance, by using k=*maximum load* we can eliminate the gap. The conjecture is that for *every fixed graph,* we can eliminate the gap *for all possible instances.*

While trying to settle the conjecture we observed that it was not clear that other, apparently natural questions about the gap were even decidable. For example, given a graph G, is there a request that will produce a (pointwise) gap?

While for any given instance it is clearly decidable whether the minimal number of colors equals the maximum load, it is not clear that it is decidable whether there is an instance that produces a gap. A path may appear an arbitrary number of times in an instance, and there is no obvious argument why there should be a computable bound N on the multiplicities of paths, such that a gap, if it exists, will be present for instances with all the requests less than N.

The existence of an (asymptotic) gap (see precise definition at the end of section) is even more problematic, since it depends on the existence of an infinite sequence of instances, and an argument that the sequence of pointwise gaps converges to a value $g > 1$.

The main results of this paper are:

Decidability of Gap Existence. Both pointwise and asymptotic gaps have clean combinatorial characterizations (Theorem 4).

Gap Computation. The asymptotic gap is computable (Theorem 7).

Multiple Fiber Conjecture. is now the Multiple fiber Theorem (Theorem 14).

In the process of proving these theorems, we develop a decomposition theory for sets of paths on a fixed graph into *prime configurations*, that, we believe is of

independent interest. Prime configurations provide clean proofs of the kind of compactness results we need to prove, and, we hope that they will be useful to tackle other problems in the area.

We present more precise definitions below. A (single-fiber) *all-optical network* is a graph $G = (V, E)$. The vertices $V = \{1, \ldots, n\}$ are the *nodes* of the network and $E \subseteq \{(i, j) \mid 1 \leq i, j \leq n\}$ is the set of node-to-node connections available in the network. A k-fiber network is a multigraph, where, for each pair u, v of vertices, either there is no edge (u, v) or there are exactly k parallel (u, v) edges. From now on we will use "wavelengths" and "colors" interchangeably. A legal *wavelength assignment* of cardinality m for a set I of paths on a graph is a map W from I to $[1, \ldots, m]$ (the *colors*), such that if two elements $p, q \in I$ *share an edge* then $W(p) \neq W(q)$, i.e., they are given distinct colors. This defines two variant models, depending on the interpretation of "...sharing an edge...": in the *directed model* both (directed) paths contain the same directed edge, while in the *undirected model* both (undirected) paths share an edge. In both models we can consider graphs with multiple edges: we assume that we have a fixed number k of parallel edges between any pair of connected vertices. The *wavelength assignment problem* can be formulated as follows:

Given a graph G and a set of paths I on G, find a legal coloring assignment W for I such that the cardinality of W is the minimum among all possible legal coloring assignments W for I.

We will denote this quantity by $W(G, I)$. We will omit the variable G if it is clear from the context. The technology we are modeling is optical fiber connections, with reserved circuits between points. This is a possible implementation of high bandwidth long-term services like on-demand video, remote medical imaging, distributed supercomputing, etc. The vertices of the multigraph correspond to nodes of the communication network, the edges to optical fibers. For $k > 1$ we assume that there are k fibers between any pair of connected nodes (and each node has an optical switch that connects any incoming fiber to any outgoing fiber). Colors correspond to different wavelengths that can be transmitted simultaneously by a fiber.

The single-fiber model has been exhaustively studied (see [2] for a survey: the bibliography has a number of more recent papers): for the k-fiber variant, see [15] [12].

Number of colors and network congestion. The *maximum load* $L(I)$, defined as follows. The *load* of an edge, $L(e)$ is the number of paths that contain e, and $L(I)$ is the maximum, over all edges e of $L(e)$. (In the directed model we consider both the path and the edge as directed).

A *legal k-wavelength assignment* W of cardinality m for a set of paths I is a map from I to $[1, \ldots, m]$ such that if $k + 1$ elements $p_1, \ldots, p_{k+1} \in I$ *share an edge* then there exist $1 \leq i, j \leq k+1$ such that $W(p_i) \neq W(p_j)$ (k is the number of parallel fibers: in a legal assignment no two connections sharing a fiber have the same color).

Let I be a set of paths on a graph $G = (V, E)$. We define the *conflict graph* $G_c = (V_c, E_c)$ for I, and G as having vertices $V_c = I$ and edges $E_c = \{(p_i, p_j) \mid p_i$ and p_j share an edge of $G\}$.

We denote by $W(I, G, k)$ the cardinality of the best possible legal k-coloring assignment for I. It is easy to verify that $W(I, G, 1)$ is equal to the chromatic number of G_c.

Let $L(I, G, \alpha)$, the *load of* α be the maximum number of paths of I sharing the edge α. Let $L(I, G)$ be the maximum of $L(I, G, \alpha)$ over all the edges α of G. It is easy to verify that $W(I, G, k) \geq \lceil \frac{1}{k} L(I, G) \rceil$.

In the 1-fiber case (k=1) $L(I, G)$ is called the *congestion* of the network. Similarly, we will call the quantity $\lceil \frac{1}{k} L(I, G) \rceil$ the k-congestion (or, when k is clear from the context) simply, *congestion*.

We say that a set of paths I has a *point-wise k-gap g* on G if $W(I, G, k) > \lceil \frac{1}{k} L(I, G) \rceil$, i.e. the number of colors needed is greater than the congestion.

Fix a graph G. Let $S = \{I_i\}_{i \in N}$ be an increasing sequence (in the "multiset inclusion" ordering) of multisets of paths on G. We say that S produces an *asymptotic* k-gap g on G if and only if

$$\text{for every } i \geq 1 : \qquad \frac{W(I_i, G, k)}{\lceil \frac{1}{k} L(I_i, G) \rceil} \geq g.$$

We denote by $Gap(S, G, k)$ the maximum k-gap that S produces on G. We define the k-gap of G, denoted by $Gap(G, k)$, as the supremum of $Gap(S, G, k)$ taken over all possible sequences S. Again, we will omit k when its value is clear from the context. We define $N(G)$ as the minimum k such that $Gap(G, k) = 1$, if such a k exists. Sometimes, we will say that there is no gap or that there is a gap if g is 1 or greater than 1, respectively. In the sequel, for shortness, we will use 'set' also to denote 'multiset'.

2 Prime Sets of Paths

We characterize potential minimal instances that have gaps, and then show that the maximum load of these instances can be bounded. We shall use the terminology "set of paths" and "instance" interchangeably. For simplicity, in this conference version of the paper we specialize the development of this section to single fiber networks.

Definition 1 (Prime set of paths). *A set S of paths is prime if it cannot be partitioned into two nonempty sets P and Q with $L(S) = L(P) + L(Q)$.*

As an example, consider again the clockwise oriented triangle and the instance consisting of the requests { [1,3], [2,1], [3,2]}. It is a prime set of paths. Otherwise, without loss of generality, one of the nonempty subsets of paths contains a single path, say [1,3]. It has load 1, while the other subset has load 2, which is the same as the load of the instance. In the next theorem we show that testing primality is an NP-complete problem even if we restrict to simple networks, e.g., 2 dimensional meshes and to set of paths of load 3 (while path set primality testing is in P for instances of load 2).

Theorem 2. *Testing primality of sets of paths of load at most 2 for a general 1-fiber network is in P while testing primality of sets of paths of load 3 for 2 dimensional 1-fiber grid is NP-complete.*

The proof will be presented in the full journal version.

Definition 3 (Complete set of paths). *A set S of paths is complete if for every edge $e \in E$ we have $L(e) = L(S)$.*

Every prime configuration can be extended to a complete prime configuration, as follows. For each edge $e = (i, j)$ such that $L(e) = k < L(P)$ add to P the path $[i, j]$ (of unit length) with multiplicity $L(P) - k$. This will not destroy the primality of the set: for every partition U, V of P we do not have $L(U) + L(V) = L(P)$. In fact we must have $L(U) + L(V) > L(P)$, since the construction adds unit paths that do not increase $L(P)$, and adding paths cannot decrease $L(U)$ or $L(V)$. Hence the inequality holds for S.

Observe that the minimal instance I that produces a gap (minimal in the sense that it has the smallest possible load $L(I)$) must be prime.

Our strategy will be to decompose any load into a collection of prime paths (with multiplicities). While the decomposition is not, in general, unique, it is a useful tool. In particular, we have the following theorem:

Theorem 4. *For any graph G the number of prime configurations is finite.*

Proof. This is an immediate consequence of the fact that there is a bound on the maximum load of a prime set of paths. This fact is stated formally below as Theorem 5; its proof is the main result of this section.

Theorem 5. *There exists a bound L_G (which depends on the size of G) such that the load of every prime set of paths on G is at most L_G.*

Proof. From now on we work with the complete set S of paths. It can be represented by m nonnegative integers $n^1, \ldots, n^m$, where m is the number of distinct paths on G, and n^i is the number of occurrences of the i-th path p_i in S.

Assume by contradiction, that there is no bound on the maximum load of a prime set of paths. Then there exists an infinite sequence $\{S_i\}_{i \in N}$ of prime sets of paths with increasing load. Using the claim above, we may assume that each S_i is complete and prime.

Consider the representation of S_i. For each coordinate j, $j = 1, \ldots, m$ consider the infinite sequence

$$N^j = n_1^j, n_2^j, \ldots, n_k^j, \ldots.$$

For each j there are two possibilities: either N^j is bounded or not. At least one of the sequences must be unbounded, since the load is. Let M be a bound on the values of the bounded sequences, and assume that t of the m sequences are bounded. By taking subsequences, we can assume that each of the unbounded sequences is monotone increasing.

Now consider the $(M+1)^t$ possible fixed values for the t bounded sequences. For each such $t-$tuple, consider the subsequence corresponding to that fixed value. At least one of them is infinite, with monotonically increasing sequences in the unbounded coordinates. Let S_q and S_p, $p > q$ be two elements of this sequence. We have $L(S_p) = L(S_p - S_q) + L(S_q)$ contradicting the hypothesis.

Note that the argument above is nonconstructive (but, it already suffices to prove the Multiple Fiber Conjecture–see Section 4). In the next theorem we show that it is possible to compute the constant L_G of Theorem 2.

Theorem 6. *Let P be a prime set of paths. Then either P is the prime set of paths with maximum load or there exists a prime set Q of paths such that $L(Q) \leq |E|\, L(P)$.*

Proof. Let t be any path of a prime set T. Since T is prime, we have that $L(T \setminus \{t\}) = L(T)$. We remove from T as many paths as we can maintaining primality. At the end of this pruning procedure we get a new prime configuration T' with the additional property that removing any path from T' yields a nonprime configuration. Moreover, $L(T') = L(T)$. We call such a set of paths *critical*.

The basic idea of this proof is that removing a path q from a critical set of paths we obtain a nonprime set of paths that is partitionable into no more than $length(q)$ (number of edges in q) prime factors. The claim yields the theorem, since $length(q) \leq |E|$. We prove the claim.

Assume there is a prime Q with $L(Q) > L(P)$, and choose Q to have mimimum load (among prime configurations with load greater than $L(P)$). Without loss of generality assume Q to be critical. Assume by contradiction that $L(Q) > |E|\, L(P)$. Let q be any path of Q. Let $Q' = Q \setminus \{q\}$. Since Q is critical Q' is not prime and therefore it can be partitioned into α prime sets F_i. Since by hypothesis $L(F_i) \leq L(P)$ we have that $\alpha \geq |E| + 1$. Taking advantage of the fact that $L(Q) = L(Q')$, one can prove that it is always possible to select a subset A of at most $length(q)$ factors F_i that satisfies the following property

$$L\left(\bigcup_{F_i \in A} F_i \cup \{q\}\right) + L\left(\bigcup_{F_i \notin A} F_i\right) = L(Q).$$

Since $length(q) < |E| + 1$, $\bigcup_{F_i \notin A} F_i$ is not empty, and therefore Q is not prime which is a contradiction.

In the next section, we consider fixed multigraphs. As we argued previously, it is not clear whether a gap exists (it might be the case that gaps appear only for requests that have paths with high multiplicity, and it is unclear, a priori, that there must be a bound on the multiplicities). Moreover, it is conceivable that a given load produces a gap (a "pointwise gap"), but there is no asymptotic gap. Finally, while there must exist a proof that a pointwise gap occurs (exhibit the load), there is no straightforward argument that we know of, that an asymptotic gap must be provable. We will show that, in fact, all these problems have finiteness properties that ensure that the gap is computable.

3 The Gap Is Computable

Again, our approach is to obtain a minimal set of 'canonical' gap-producing instances, claim that they are in some sense responsible for the maximal gap, and compute the gap as the result of repetitions of a 'bad' instance. Not surprisingly, the canonical instances will be prime configurations.

Theorem 7. *If G with k fibers produces an asymptotic gap g then there is a sequence of instances consisting of multiple copies of the same prime set of paths that produces the asymptotic gap g.*

Proof. We give a brief outline of the proof strategy. Every set S of paths can partitioned into a collection of prime sets (the same prime set may appear multiple times). Again, first make a set of paths complete by adding length 1 paths if necessary: this will change neither the load nor the number of wavelengths. The resulting set is either prime (and we are done), or it can be decomposed into two sets A and B with $L(S) = L(A) + L(B)$. Both A and B are complete, with strictly smaller loads, and we proceed inductively.

Prime sets that do not produce a pointwise gap do not influence the asymptotic gap, and can be dropped from the decomposition. It is not hard to see that the gap produced by the entire sequence cannot be bigger than the maximum gap produced by taking multiple copies of one of the prime sets obtained from the initial partition.

While the theorem above provides a characterization of a canonical set of paths that will produce maximal asymptotic gap, the result does not allow us to compute the gap: we only know that the canonical set consists of certain multiples of a single prime set. We need to determine this multiplicity (again, there seems to be no obvious argument that would yield a bound on the multiplicity).

For the moment we prove that the gap produced by some such canonical sets is computable.

Theorem 8. *The gap produced by multiple copies of a given set of paths is computable*

Proof. Let $P = (n_1, \ldots, n_m)$ be a prime set of paths. We say that a set P of paths is *full* if adding an extra path to P increases its load. For each full set P_i of paths with $L(P_i) = k$ (assume we have n sets of this type) we define a real variable X_i. Let c_{ij} be the number of copies of path p_j belonging to P_i and n_i be the number of copies of path p_i belonging to P. Let X_i^* be the optimal solution to the linear programming problem:

$$Minimize \sum_{i=1}^{n} X_i$$

subject to

$$\begin{aligned} c_{11}X_1 + \cdots + c_{n1}X_n &\geq n_1 \\ \vdots \quad &\vdots \\ c_{1m}X_1 + \cdots + c_{nm}X_n &\geq n_m \end{aligned}$$

If all X_i^*s were integers clearly we would have found the optimal solution to the wavelength assignment problem associated to P and to multiple copies of P as well. Moreover, the gap produced by P would be $\frac{\sum_{i=1}^{n} X_i}{L(P)}$. We call this quantity *fractional gap*. Actually, X_i^*s are rational numbers. As a consequence, we have that there exists a natural number c such that the optimal solution associated to c copies of P consists of integer values. The same can be said if we consider ic copies $i \in N$ of P instead of a single copy.

These results extend to the multifiber case.

Theorem 9. *Given a network G with k fibers per link it is possible to compute $Gap(G, k)$.*

Proof. $Gap(G, k)$ is equal to the maximum fractional gap given by a prime set of paths.

3.1 Gap in 1-Fiber Networks

In this section we prove explicit implications among the existence of pointwise gaps, prime sets, and asymptotic gaps in the case of 1-fiber networks, delaying the general multifiber proofs to a complete version.

Lemma 10. *Let G be any 1-fiber network. There exists a prime set P of paths on G with load $L(P)$ greater than 1 if and only if there exists a set Q of paths on G producing a pointwise gap.*

Proof. Let P be a prime set of paths of load $L(P)$. Since P is prime, it cannot be partitioned into $L(P)$ monochromatic sets of paths, and therefore P has a pointwise gap. Conversely, if Q has a pointwise gap and is not prime, it can be partitioned into α prime sets. At least one of them must produce a pointwise gap (or Q would not have a gap) and then it must have load greater than 1.

Lemma 11. *Let Q be any set of paths on a graph G that produces a gap, and let mQ be the union of $m \geq 1$ copies of Q. If Q is prime then mQ has a pointwise gap.*

Proof. As usual, we assume without loss of generality that Q is complete. Assume by contradiction that mQ has no pointwise gap. Then it is possible to split mQ into $L(mQ)$ prime and complete sets F_i of monochromatic paths with load 1. Since $L(F_i) = 1$, we conclude that $F_i \subset Q$ and therefore Q is not prime. This completes the proof.

Note that
– the pointwise gap of mQ might approach 1–no (asymptotic) gap–for increasing values of m, and
– the proof of Lemma 11 cannot be trivially extended to the case k-fiber networks with $k > 1$ since we cannot say that "...we conclude that $F_i \subset Q$ ...". In fact, when k is greater than 1 it is possible to find a k-fiber network G and a prime set P of paths on G such that multiple copies of P do not produce a pointwise (and then an asymptotic) gap (details in the full paper).

Lemma 12. *P has fractional gap g (possibly $g = 1$) then there exists an integer $m \geq 1$ such that for every $i \geq 1$ imP has pointwise gap g.*

Proof. Omitted in this version.

Theorem 13. *Let G be any 1-fiber network. The three following statements are equivalent.*
1- G has an asymptotic gap,
2- there exists a prime set P of paths on G with load greater than 1,
3- there exists a set Q of paths on G that produces a pointwise gap.

Proof. Statements 2 and 3 are equivalent by Lemma 10.

It is easy to see that if there are no prime configurations with load greater than 1 then it is always possible to find a coloring for every set of paths with cardinality equal to its load (it is sufficient to assign to each prime configuration a distinct color).

We now give a brief outline of the proof that if P prime has a load greater than 1 then there is an asymptotic gap. Assume that there is no asymptotic gap. Then P has no fractional gap (otherwise multiple copies of P would produce an asymptotic gap). Using Lemma 12 we have that there exists $m \geq 1$ such that mP has no pointwise gap. Using Lemma 11 we conclude that P cannot be prime.

4 Multiple Fiber Conjecture

The multiple fiber conjecture holds.

Recall that the conjecture is that for every fixed graph G the gap can be eliminated by a fixed number $N(G)$ of multiple edges. Our proof is a direct consequence of the fact that the set of prime configurations of a fixed graph is finite, and the maximum load of a prime configuration is bounded.

Theorem 14. *Let L_G be the bound on the load in the statement of theorem 2. There is no gap in the graph G with $\alpha = \frac{L_G!}{\lfloor L_G/2 \rfloor !}$ parallel fibers for each edge.*

Proof. It suffices to ensure that there are no gaps for prime instances. It is not hard to prove that if there is no gap for an instance I in a network with m fibers, there is no gap in any network of im fibers, for $i = 1, 2, \cdots$. Consider a prime set of paths. It has load k for some $1 \leq k \leq L_G$. Clearly, there is no gap if we have a number of fibers that is a multiple of k, therefore if α is is an integer such that it is a multiple of every number between 1 and L_G and we consider the graph with α fibers per edge, there is no gap for any prime set of paths. Therefore there is no gap.

5 A Case Study: The Ring

An n-ring is a graph $G = (V, E)$ with
$V = \{x_0, \ldots, x_{n-1}\}$ and $E = \{(x_i, x_{i+1}),\ i = 0, \ldots, n-2\} \cup \{(x_{n-1}, x_0)\}$.

While in general graphs prime configurations are computationally intractable, many aspects of our decomposition theory are feasible on rings. We present these results as concrete illustration of our techniques, as a simple application, and as a means to get a better intuition. We believe that prime configurations may be useful for general graphs, possibly as a tool to obtain approximation algorithms for the wavelength assignment problem, and to get approximate lower bounds.

For ring networks there is no difference between the directed and the undirected model for the wavelength assignment problem. Every set of paths in the directed model can be partitioned into two disjoint sets of paths, C, paths routed in the clockwise direction and CC, routed in the opposite direction. Since there are no conflicts among paths belonging to C and CC (they use different directions on every edge), the original problem is equivalent to the two undirected wavelength assignment problems given by the set of requests in C and CC. For this reason, we will consider only the problem of assigning wavelengths to a set I of undirected paths on a ring.

Even if in the undirected model paths have no starting and ending nodes, in what follows we assign to each path a start node and an end node according to the clockwise orientation. So, for example, a path with endnodes x_5 and x_2 that goes through nodes x_4 and x_3 will be denoted by the pair (x_2, x_5). Using this notion of start/end nodes it makes sense to say that two paths start at the same node or end at the same node.

Theorem 15. *Let P be a complete set of paths on any n-ring. If two paths of P start at the same node then P is not prime.*

Proof. Let p be any path of P. Let x_j be the start node of p. Since P is complete there exists a path $q \in P$ such that the end node of p is equal to the start node of q. We call such a node the *successor* of p. Assume that there is another path p' starting at x_j. Since P is complete we can list all the paths of P in such a way that the ith element of the list is the successor of the $(i-1)$th element. Assume that p' has index m in the list. Then the first $m-1$ elements of the list form a complete set Q of paths. Moreover, since p' is not included in Q we conclude that P can be decomposed into two complete nonempty sets namely, Q and $P \setminus Q$, contradicting the hypothesis.

The following corollary follows directly from Theorem 15.

Corollary 16. *The maximum load of a prime configuration on an n-ring is at most $n-1$.*

Moreover, we can prove the following result.

Corollary 17. *Every prime set of paths with load greater than k on a k-fiber ring produces an asymptotic gap.*

Proof. The proof of this corollary is along the lines of the proof of Theorem 13. Theorem 13 cannot be extended to general k-fiber networks since in general it is not true that prime sets of paths contain at most one copy of each path. However, for ring networks Theorem 15 ensures that prime sets satisfy this property, so the proof of Theorem 15 works also for k-fiber rings for any k.

The following lemma is used to compute the number of fibers that are necessary to have no asymptotic gaps on n-ring.

Lemma 18. *Let G be an n-ring. For every $m = 1, \ldots, n-1$ there exists a prime set of paths of load m.*

Proof. Consider the set P of all the paths of length $n-1$. All these paths start at distinct nodes, so P is prime (Theorem 15). It is easy to verify that $L(P) = n-1$. In order to get a set of paths of load $m < n-1$ it is sufficient not to consider $n-1-m$ nodes and repeat the construction above for load $n-1$.

Lemma 19. *Let G be a k-fiber n-ring. Let P be a prime set with load $m < k$ and iP be the set of paths consisting of i copies of P. Assume that k is not a multiple of m. Then every subset X of iP with load k is not complete.*

Proof. As usual we assume that P is complete. Assume by contradiction that there exists a complete set $X \subset iP$ of load k. X can be partitioned into $h \geq 1$ prime and complete sets P'_i. Since each P'_i is prime, it does not contain paths with multiplicity greater than 1 (Theorem 15) and so $P'_i \subseteq P$. Moreover, being complete, P'_i cannot be a proper subset of P (otherwise P would not be prime.) We conclude that $P'_i = P$ for every $i = 1, \ldots h$ and therefore $X = hP$ which implies that $k = hm$, a contradiction.

Using these characterizations, we obtain a bound on the number of fibers, $N(G)$ that ensures that there is no gap on a ring G.

Theorem 20. *Let G be a k-fibers n-ring. Then $N(G) = \frac{(n-1)!}{\lfloor (n-1)/2 \rfloor !}$.*

Proof. Combining Theorem 14 and point 3 of Corollary 16 we have that $N(G) \leq \frac{(n-1)!}{\lfloor (n-1)/2 \rfloor !}$. Assume now that k is strictly smaller than $\frac{(n-1)!}{\lfloor (n-1)/2 \rfloor !}$. Then there exists an integer $m \leq n-1$ such that:
- k is not a multiple of m,
- there exists a prime set P of paths with load m on G (Lemma 18).
Assume by contradiction that using k fibers we have no asymptotic gap on G. Then there is no fractional gap for P. As a consequence we have that for infinitely many i, iP has no pointwise gap. Thus, we can partition iP (for a suitable i) into a suitable number of complete sets of paths each of them with load k, contradicting Lemma 19.

References

1. R. P. Anstee. *An algorithmic proof of Tutte's f-factor theorem.* Journal of Algorithms, 6:112–131, 1985.
2. B. Beauquier, J.-C. Bermond, L. Gargano, P. Hell, S. Perennes, and U. Vaccaro. *Graph problems arising from wavelength-routing in all-optical networks.* Proc. of Workshop on Optics in Computer Science WOCS'97.
3. N. K. Cheung, N. K., and G. Winzer. *Special issue on dense WDM networks.* Journal on Selected Areas in Communications, 8, 1990.

4. T. Erlebach and K. Jansen. *Scheduling of virtual connections in fast networks.* In Proc. of Parallel Systems and Algorithms (PASA), pages 13–32, 1996.
5. T. Erlebach and K. Jansen. *Call scheduling in trees, rings and meshes.* In Proc. of HICSS, 1997.
6. M. C. Golumbic and R. E. Jamison. *The edge intersection graphs of paths in a tree.* Journal of Combinatorial Theory, Series B, 38:8–22, 1985.
7. P. E. Green. *Fiber–optic communication networks.* Prentice–Hall, 1993.
8. I. Holyer. *The NP–completeness of edge coloring.* SIAM Journal of Computing, 10(4):718–720, 1981.
9. C. Kaklamanis, G. Persiano, T. Erlebach,and K. Jansen. *Constrained bipartite edge coloring with applications to wavelength routing.* Proc. of ICALP'97, Lecture notes in Computer Science vol. 1256:493–504, 1997.
10. R. Klasing. *Methods and problems of wavelength-routing in all-optical networks.* In Proc. of the MFCS'98 Workshop on Communication, 1998.
11. J. Kleinberg and A. Kumar. *Wavelength conversion in optical networks* In Proc. 10th ACM-SIAM Symposium on Discrete Algorithms, 1999.
12. G. Li and R. Simha *On the wavelength assignment problem in multifiber WDM star and ring networks* In Proc. IEEE INFOCOM, 2000.
13. L. Lovász. *On chromatic number of finite set-systems.* Acta Math. Acad. Sci. Hungar, 19:59–67, 1968.
14. A. D. McAulay. *Optical computer architectures.* John Wiley, 1991.
15. L. Margara and J. Simon *Wavelength assignment problem on all-optical networks with k fibers per link* Proc. of ICALP2000, Lecture notes in Computer Science vol. 1853:768–779, 2000.
16. M. Mihail, C. Kaklamanis, and S. Rao. *Efficient access to optical bandwidth—wavelength routing on directed fiber trees, rings, and trees of rings.* In Proc. of 36th IEEE-FOCS, pp. 548–557, 1995.
17. R. K. Pankaj and R. G. Gallager. *Wavelength requirements of all-optical networks.* IEEE/ACM Trans. on Networking, 3:269–280, 1995.
18. J. Petersen. *Die Theorie der Regulären Graphen.* Acta Math. 15, 193–220, 1891.
19. R. Ramaswami. *Multiwavelength lightwave networks for computer communication.* IEEE Communications Magazine, 31(2):78–88, Feb. 1993.
20. R. E. Tarjan. *Decomposition by clique separators.* Discrete Mathematics, 55(2):221–232, 1985.
21. A. Tucker. *Coloring a family of circular arcs.* SIAM Journal of Applied Mathematics, 29(3):493–502, 1975.
22. R. J. Vetter and D. H. C. Du. *Distributed computing with high-speed optical networks.* IEEE Computer, 26(2):8–18, Feb. 1993.
23. G. Wilfong and P. Winkler. *Ring routing and wavelength translation.* In Proc. of the 9th Annual ACM-SIAM Symposium on on Discrete Algorithms (SODA), pp. 333–341, 1998.

Majority Consensus and the Local Majority Rule

Nabil H. Mustafa[1] and Aleksandar Pekeč[2]

[1] Department of Computer Science,
Duke University.
nabil@cs.duke.edu
[2] The Fuqua School of Business,
Duke University.
pekec@duke.edu

Abstract. We study a rather generic communication/coordination/computation problem: in a finite network of agents, each initially having one of the two possible states, can the majority initial state be computed and agreed upon by means of local computation only? We describe the architecture of networks that are always capable of reaching the consensus on the majority initial state of its agents. In particular, we show that, for any truly local network of agents, there are instances in which the network is not capable of reaching such consensus. Thus, every truly local computation approach that requires reaching consensus is not failure-free.

1 Introduction

Attempting to solve a complex problem by a simultaneous coordinated activity of local agents is an idea that arises naturally in a variety of contexts. For example, this idea is fundamental in frameworks as diverse as distributed computing and neural networks. While methods of local computation and decision-making are often effective in dealing with complex tasks, the successful implementation of such methods often raises a new breed of problems related to coordination and communication of local agents.

We study a discrete time, memoryless, synchronous dynamic process and call it *local majority process* on (a finite network) G. Informally, the vertices of a graph $G = (V, E)$ represent the agents and the edges of G represent all (bidirectional) communication links between pairs of agents. Initially, at time $t = 0$, each agent is in one of the two possible states, e.g., colored red or blue (voted Yes or No, having value 0 or 1, ...). Then the local majority rule is applied synchronously and iteratively: an agent has different colors at time t and $t + 1$ if and only if the agent's color at time t is not a majority color in the agent's neighborhood in G at time t. A precise formulation of the model will be given in the next section.

The local majority process (and some of its natural extensions) has been studied in frameworks as diverse as social influence [PS83, PT86a, PT86b] and

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 530–542, 2001.

neural networks [GO81,GO80,Gol86,GM90]. Recently, the local majority process has reappeared (under the name *polling process*) in several papers motivated by certain distributed computing problems [Pel98,Ber99,FLLS98,FLL+99, Has98,HP99,HP00,LPS99,NIY99,NIY00].

A natural question to ask is when does the local majority process ensure that all agents reach a consensus on the initial majority state? We will say that G is a *majority consensus computer* (m.c.c.) if, for any set of initial states (there are $2^{|V|}$ such sets), the local majority process simultaneously brings all agents into the state that was the initial majority state. Note that, according to the local majority process, once all agents are in the same state, no agent will change its state ever after. All of the recent papers dealing with the local majority process and its modifications [Pel98,Ber99,FLLS98,FLL+99,Has98,HP99,HP00,LPS99, NIY99,NIY00] investigated how badly could the local majority process (and its variations) miscalculate the initial majority (on a specific class of graphs)[1]. In contrast to these results, we are interested in G which are immune to miscalculations in the local majority process, i.e., the focus of this paper are majority consensus computers and investigation of their structure.

Since being a majority consensus computer is seemingly a very strong property, one would expect that a sort of an impossibility theorem holds. As will be shown, the situation is not that simple and the full characterization of m.c.c.'s remains an open problem. However, our results demonstrate in several ways that the non-locality is inherent property of every m.c.c.. Thus, reaching consensus on the majority is a truly non-local task in the sense that a most natural local computation procedure is failure-free only if computing local majority is essentially as complex as computing global majority.

2 Majority Consensus Computers

A standard graph theoretic notation is used throughout the paper. $G = (V, E)$ denotes an undirected, simple, finite graph G with the vertex set V, $|V| = n$, and the edge set E (i.e. $E \subseteq \{S \subseteq V : |S| = 2\}$). The *neighborhood* of vertex v in set $S \subseteq V$ is the set of all neighbors of v that are in S, $N_S(v) := \{s \in S : \{v, s\} \in E\}$, and the *degree* of v in S is $deg_S(v) = |N_S(v)|$ (if $S = V$, the subscript is omitted). Min and max degree in G are denoted by $\Delta(G) = \max\{deg(v) : v \in V\}$ and $\delta(G) = \min\{deg(v) : v \in V\}$. Some non-standard terminology: a vertex v is a *master* if $deg(v) = n - 1$ (i.e., v is adjacent to every other vertex). We also say that v is a *k-master* if $deg(v) = n - 1 - k$ (i.e., v is adjacent to all but k other vertices). Note that 0-master and master are equivalent notions, and we will use them interchangeably throughout the rest of the text.

In our model all agents and communication links in the system are represented by a graph G in a natural way. That is, the vertices of G are in a

[1] For example, Berger [Ber99] has shown that for every n there exists a G on at least n vertices and the set of states such that only 18 vertices are in one state and the rest are in the other, yet the local majority process forces all vertices to simultaneously end up in the initial minority state.

one-to-one correspondence with the agents and the edges of G correspond to adjacency relations among the agents.

A *coloring* of the graph G, $c^t : V \to \{0,1\}$ defines an assignment of binary values (colors) to the vertices of G at time t. We use the notation $c_v^t := c^t(v)$ to denote the color of a vertex v at time t. The notation $sum(c^t) := \sum_{v \in V} c_v^t$ will also be useful. A color which is assigned to more than $|V|/2$ vertices at a time t is called the *majority color* of the coloring c^t and denoted by $maj(c^t)$. Thus, $maj(c^t) = 1$ if and only if $sum(c^t) > n/2$ and $maj(c^t) = 0$ if and only if $sum(c^t) < n/2$. Note that $maj(c^t)$ is not defined if $|V|$ is even and c^t defines an equipartition of V, i.e., if $sum(c^t) = n/2$. A coloring c^t is a *consensus* if it is constant, i.e. if all the vertices of G have the same binary values (colors). Thus, c^t is a consensus if and only if $c_v^t = maj(c^t)$ for all $v \in V$. We will sometimes abuse the notation and write $c^t = 0$ or $c^t = 1$ for consensus in color 0 and 1, respectively. Another abuse of notation is $(1 - c^t)$ denoting the coloring obtained from c^t by changing the color of every vertex, i.e., for every $v \in V$ and coloring c^t, $(1 - c^t)(v) = 1 - c^t(v)$.

The main object of our study is the *local majority process* $LMP(G, c^0)$, a discrete time process on G that is based on the iterative application of the local majority rule. The process is completely defined by G and the *initial coloring* c^0. For every $t = 0, 1, 2, \ldots$, the coloring c^{t+1} is derived by applying the *local majority rule* on $N(v)$ for each vertex in G:

$$c_v^{t+1} = \begin{cases} c_v^t & \text{if } |\{w \in N(v) : c_w^t = c_v^t\}| \geq |N(v)|/2 \\ 1 - c_v^t & \text{if } |\{w \in N(v) : c_w^t \neq c_v^t\}| > |N(v)|/2 \end{cases} \quad (1)$$

The local majority rule simply states that, at the next discrete time step, the color assigned to a vertex v will be the color of the majority of its neighbors. Note that an even degree vertex will retain its color whenever exactly half (or more) of its neighbors have the same color. The above rule also implies that the local majority rule is executed simultaneously for all the vertices. We say that there is a *majority switch* at time $(t + 1)$ if $maj(c^t) \neq maj(c^{t+1})$.

Note that if c^t is a consensus, then $c^{t+k} = c^t$ for all positive integers k. If, for some positive integer t, c^t is a consensus, then we say that G *reaches consensus* for c^0. If G reaches consensus c^t for c^0 and $c^t = maj(c^0)$, then we say that the $LMP(G, c^0)$ *correctly computes the initial majority* and that G admits a *majority consensus* for the initial coloring c^0. A graph G is a *majority consensus computer* (or a *m.c.c.* in short) if, for every c^0 (there are 2^n such colorings), $LMP(G, c^0)$ correctly computes the initial majority. In other words, G is a m.c.c. if G admits majority consensus for all of the 2^n possible initial colorings. Note that for every graph with even number of vertices there exists a c^0 where $maj(c^0)$ is not defined. Thus, G can be a majority consensus computer only if it has an odd number of vertices. **Throughout the rest of the paper we assume that n is odd.**

Our first observation about majority consensus computers is the following proposition.[2]

[2] The proofs omitted throughout the paper can be found in [MP00].

Proposition 1. *Let G be a m.c.c. and let c^0 be an initial coloring of G. Then there are no majority switches for $LMP(G, c^0)$, i.e., $maj(c^t) = maj(c^0)$ for $t = 0, 1, 2, \ldots$.*

Next note that there are only 2^n possible colorings and c^{t+1} is a function of G and c^t, thus the sequence $c^0, c^1, c^2, \ldots$ must become periodic, i.e., there exists positive integers t_0 and k such that $c^{t+k} = c^t$ for every $t \geq t_0$. Obviously, the period k and t_0 are not larger than 2^n. Somewhat surprisingly, the period can be only one or two and there exists t_0 smaller than $|E|$.

Theorem 1. *Consider the sequence $c^0, c^1, c^2, \ldots$ defined by the local majority process on G with initial coloring c^0, $LMP(G, c^0)$. Then there exists $t_0 < |E|$ such that $c^t = c^{t+2}$ for every $t \geq t_0$.*[3]

Many of our results will be based on the *"period is at most two"* property. Next we show that a monotonicity property with respect to the structure of the coloring holds in the local majority process.

Lemma 1. *Let $V_i(c^t) = \{v \in V : c_v^t = i\}$, $i = 0, 1$, where c^t is a coloring of $G = (V, E)$. If there exists $i \in \{0, 1\}$ and colorings c^t and $d^{t'}$ such that $V_i(c^t) \subseteq V_i(d^{t'})$ then $V_i(c^{t+k}) \subseteq V_i(d^{t'+k})$ for $k = 0, 1, 2, \ldots$.*

According to the definition, in order to check whether G is a majority consensus computer, one would have to check whether G admits majority consensus for all 2^n possible initial colorings c^0. However, because of the monotonicity property described in Lemma 1, it suffices to consider only colorings c^0 such that $sum(c^0) = (n+1)/2$. (There are $\binom{n}{(n+1)/2} = O(2^n/\sqrt{n})$ such colorings).

Theorem 2. *G is a majority consensus computer if and only if G admits majority consensus for any c^0 such that $sum(c^0) = (n+1)/2$.*

Remark. Unfortunately, it is not true that adding an edge to or deleting an edge from a majority consensus computer G preserves the property *"majority consensus computer"*. For example, consider $(K_n)^c \subset K_n \setminus P_{n-1} \subset K_n \setminus P_{(n+1)/2} \subset K_n$ where n is odd. It is not difficult to show that K_n and $K_n \setminus P_{n-1}$ are m.c.c., while $(K_n)^c$ and $K_n \setminus P_{(n+1)/2}$ are not (see [MP00]).

We close this section by showing that masters in G compute majority instantly, i.e., the color of a master at time $t+1$ is $maj(c^t)$. (Larger the difference between the majority and minority color of c^t, smaller degree of v is needed to ensure $c_v^{t+1} = maj(c^t)$.)

Proposition 2. *If v is a master in G, then $c_v^{t+1} = maj(c^t)$. More generally, if v is a k-master in G and $|sum(c^t) - n/2| \geq (k+1)/2$, then $c_v^{t+1} = maj(c^t)$.*

[3] This theorem is a straightforward consequence of a much more general result that can be found in, e.g., [GM90] (various variations and extensions can be found in this rather comprehensive collection of results related to dynamic behavior of neural and automata networks); sufficient conditions for the property in the case of LMP on infinite graphs were studied in [Mor94b,Mor94a,Mor95].

3 Structural Properties

Let's start by presenting a class of graphs that are m.c.c. and a class of graphs that are not m.c.c..

Proposition 3.
(a) *A graph G with more than $n/2$ masters is a majority consensus computer.*
(b) *A graph G with exactly $(n-1)/2$ masters is not a majority consensus computer.*

Next we give a characterization of majority consensus computers which indicates a way towards a static representation in the form of existence of a particular partition of the vertices of G.

Theorem 3. *G is* **not** *a majority consensus computer if and only if at least one of the following holds:*
(a) *There exists c^0 such that $maj(c^0) \neq maj(c^1)$*
(b) *There exists a partition of V into four sets A_0, A_1, B_0, B_1 satisfying*

1. $|B_0||B_1| = 0 \Rightarrow |A_0||A_1| \geq 1$,
2. *For every $v \in A_i$, $i = 0, 1$: $deg_{A_i}(v) - deg_{A_{1-i}}(v) \geq |deg_{B_i}(v) - deg_{B_{1-i}}(v)|$*
3. *For every $v \in B_i$, $i = 0, 1$: $deg_{B_{1-i}}(v) - deg_{B_i}(v) > |deg_{A_i}(v) - deg_{A_{1-i}}(v)|$*

Proof. Suppose G is not a majority consensus computer. If G admits a consensus for every possible initial coloring c^0, there must exist d^0 for which G does not admit a majority consensus, i.e., there exists d^0 and t such that d^t is a consensus and $maj(d^0) \neq maj(d^t)$. Obviously, in the sequence $d^0, d^1, \ldots, d^t$, there exists $t' < t$ such that $maj(d^{t'}) \neq maj(d^{t'+1})$. Thus, (a) holds for $c^0 := d^{t'}$.

Thus, we may assume that there exists c^0 for which G does not admit a consensus. By Theorem 1 there exists t such that $c^t = c^{t+2}$. For $i = 0, 1$ define $A_i := \{v \in V : i = c_v^t = c_v^{t+1}\}$ and $B_i := \{v \in V : i = c_v^t \neq c_v^{t+1}\}$. Note that A_0, A_1, B_0, B_1 partition V and that *1.* must hold since neither c^t nor c^{t+1} is a consensus. Since for every $v \in A_i$, $c_v^t = c_v^{t+1}$, $deg_{A_i}(v) + deg_{B_i}(v) \geq deg_{A_{1-i}}(v) + deg_{B_{1-i}}(v)$. Similarly, for every $v \in A_i$, $c_v^{t+1} = c_v^{t+2}$ implies (because $\{w : c_w^{t+1} = i\} = A_i \cup B_{1-i}$) $deg_{A_i}(v) + deg_{B_{1-i}}(v) \geq deg_{A_{1-i}}(v) + deg_{B_i}(v)$. These two inequalities imply *2.* In the same manner, it follows that for every $v \in B_i$, $c_v^t \neq c_v^{t+1}$ implies $deg_{A_{1-i}}(v) + deg_{B_{1-i}}(v) > deg_{A_i}(v) + deg_{B_i}(v)$ and that $c_v^{t+1} \neq c_v^{t+2}$ implies $deg_{A_i}(v) + deg_{B_{1-i}}(v) > deg_{A_{1-i}}(v) + deg_{B_i}(v)$. Hence, *3.* follows from these two inequalities.

The converse is straightforward to verify. (If (b) holds, set $c_v^0 = i$ for $v \in A_i \cup B_i$.) □

The last theorem indicates that majority consensus computers are highly connected graphs. For example, it follows in a straightforward manner that if a graph is bipartite or disconnected, then it is not a majority consensus computer. Furthermore, it can be shown that any majority consensus computer has trivial min-cuts, and no unique max-cuts. The following theorem and its corollary provide another confirmation of this claim.

Theorem 4. *Let G be a majority consensus computer. Then for every $v \in V$*

$$|\bigcup_{w \in N(v)} N(w)| \geq n/2. \tag{2}$$

Proof. First note that we can assume that G is connected and that $n > 2$.

Suppose (2) does not hold for some $v \in V$. Let $u \in V$ be a vertex of the minimum degree among all vertices v for which (2) is violated. Let c^0 be such that $c^0_v = 1$ for every $v \in \bigcup_{w \in N(u)} N(w)$ and such that $sum(c^0) = (n+1)/2$. Note that $c^0_u = 1$ and that $maj(c^0) = 1$. Let d^0 be such that $d^0_v \neq c^0_v$ if and only if $v = u$ (i.e., the only difference between c^0 and d^0 is in the color of u). Note that $sum(d^0) = (n-1)/2$ and thus

$$maj(d^0) = 0 \neq 1 = maj(c^0). \tag{3}$$

Observe that for all $v \notin N(v) \cup \{u\}$, $w \in N(v) \Rightarrow c^0_w = d^0_w$, and hence $c^1_v = d^1_v$. Further observe that $c^1_u = d^1_u = 1$ because the color of all neighbors of u is 1 in both c^0 and d^0 (and u has at least one neighbor since G is connected). Finally observe that by the choice of u and the fact that G is connected and $n > 2$, $deg(v) \geq 2$ for all $v \in N(u)$. Since the color of all neighbors of v other than u is 1 in both c^0 and d^0, it follows that $c^1_v = d^1_v$ for $v \in N(u)$. Hence, $c^1 = d^1$ and thus, because of (3), either $maj(c^1) \neq maj(c^0)$ or $maj(d^1) \neq maj(d^0)$. In either case, it follows from Proposition 1 that G is not a majority consensus computer. □

The theorem shows that majority consensus computers are nowhere truly local since the second neighborhood of any vertex contains a majority of the vertices of V. Hence, the local majority process always reaches a consensus on the initial majority color only if the local majority rule is **nowhere** local. Hence, the theorem can be viewed as a sort of an impossibility result.

Corollary 1. *If G is a majority consensus computer then $diam(G) \leq 4$, i.e., the length of the shortest path between any two vertices is at most* 4.

Exhaustive computer aided search confirmed that $diam(G) \leq 2$ for every majority consensus computer on at most 13 vertices. We conjecture that a much stronger statement is true (also confirmed to hold for $n \leq 13$ by an exhaustive search method).

Master Conjecture. Every majority consensus computer contains a master.

This is a rather strong conjecture because it implies that a necessary condition for reaching majority consensus is the existence of a vertex connected to all the other vertices, thereby annihilating any notion of local computation.

The master conjecture also has interesting implications. For example, it can be shown that any majority consensus computer with $(n-i)/2$ master vertices has minimum degree at least $(n-i)$. Thus the minimum degree of every vertex is strongly related to the number of master vertices in the graph.

4 Master Conjecture for Highly Connected Graphs

In this section we show that the master conjecture holds for graphs G with $\delta(G) \geq n-3$. Note that, intuitively, such graphs should be considered as prime candidates for a counterexample to the conjecture since all of the vertices in these graphs are either masters or very close to being masters (i.e., 0-masters, 1-masters, or 2-masters).

A direct consequence of Proposition 2 is that the only colorings c^0 for which G might not admit a majority consensus are the tight ones, i.e., c^0 such that $sum(c^0) = (n+1)/2$. (The case $sum(c^0) = (n-1)/2$ is symmetric).

Proposition 4. *If $\delta(G) \geq n-3$, then G admits majority consensus for every c^0 such that $sum(c^0) \geq (n+3)/2$.*

If $\delta(G) \geq n-3$, then G^c has a very simple structure since $\Delta(G^c) = (n-1) - \delta(G) \leq (n-1) - (n-3) = 2$. In other words, a connected component of G^c is a single vertex, a path, or a cycle. The decomposition of G^c into its connected components $H_1 = (V_1, E_1^c)$, $H_2 = (V_2, E_2^c)$, ..., $H_m = (V_m, E_m^c)$,[4] will be used throughout this section and we will often abuse the notation and identify $V(H)$ with H whenever such notation will be unambiguous (e.g., we will often say that the connected components of G^c define a partition of V).

Another convenient property of G with $\delta(G) \geq n-3$ is that every vertex in G is either a master, a 1-master, or a 2-master. Thus, the following lemma gives a complete boolean formula representation of local changes for colorings c^t with $sum(c^t) = (n+1)/2$.

Lemma 2. *Let c^t be such that $sum(c^t) = (n+1)/2$.*
(a) *If v is a master, then $c_v^{t+1} = 1$.*
(b) *If v is a 1-master, then $c_v^{t+1} = 1 - c_v^t c_w^t$ where w is the unique vertex not adjacent to v.*
(c) *If v is a 2-master, then $c_v^{t+1} = 1 - c_u^t c_w^t$ where u and w are the two vertices not adjacent to v.*

Proof. If v is a 2-master then $V \setminus N(v) = \{v, u, w\}$, so

$$|\{u \in N(v) : c_u^t = 1\}| = \frac{n+1}{2} - c_v^t - c_u^t - c_w^t$$

First suppose $c_v^t = 0$. Then, $c_v^{t+1} = 0$ if and only if $|\{u \in N(v) : c_u^t = 1\}| \leq |N(v)|/2 = (n-3)/2$ and this is true if and only if $c_u^t = c_w^t = 1$. Thus, (c) holds if $c_v^t = 0$. Finally, suppose $c_v^t = 1$. Then, $c_v^{t+1} = 0$ if and only if $|\{u \in N(v) : c_u^t = 1\}| < |N(v)|/2 = (n-3)/2$ and, again, this is true if and only if $c_u^t = c_w^t = 1$. Thus, (c) also holds if $c_v^t = 1$. The proof of (b) is similar, while(a) is obvious. □

This lemma allows us to track action of the local majority process on G. We define an *auxiliary graph* $AG = (V, E(AG))$. The edges of AG are defined

[4] In other words, V_i, $i = 1, \ldots m$ are pairwise disjoint, $V_1 \cup \ldots \cup V_m = V$, and $E_1^c \cup \ldots \cup E_m^c = E^c$

using the formulas from (b) and (c) from the lemma: $E(AG)=\{\{v,w\}:d_G(v)=n-2,\{v,w\}\notin E(G)\}\cup\{\{u,w\}:\exists v, d_G(v)=n-3;\{v,u\},\{v,w\}\notin E(G)\}$. Thus, $E(AG)$ is in one to one correspondence with the set of all vertices of G which are not masters. Note that AG has a rather simple structure: all of its connected components are cycles, each corresponding to a connected component of G^c as follows (this is a direct consequence of the definition of AG):

If a connected component $H \subset G^c$ is a path, say $v_1, v_2, \ldots, v_l$ (i.e., $\{v_i, v_{i+1}\} \in E(G^c)$, $i = 1, \ldots, l-1$), then $V(H)$ defines a cycle C_H that is a connected component in AG: If l is even, the adjacent vertices in C_H are $v_1, v_2, v_4, \ldots, v_{l-2}, v_l, v_{l-1}, \ldots, v_5, v_3, v_1$. If l is odd, the adjacent vertices in C_H are $v_1, v_2, v_4, \ldots, v_{l-1}, v_l, v_{l-2}, \ldots, v_5, v_3, v_1$.

If a connected component $H \subset G^c$ is an odd cycle (i.e. odd number of vertices), say $v_1, v_2, \ldots, v_{2k+1}, v_1$ (i.e., $\{v_i, v_{i+1}\} \in E(G^c)$, $i = 1, \ldots, 2k+1$, and $\{v_{2k+1}, v_1\} \in E(G^c)$), then $V(H)$ defines a cycle C_H that is a connected component in AG: $v_1, v_3, \ldots, v_{2k+1}, v_2, v_4, \ldots, v_{2k}, v_1$.

If a connected component $H \subset G^c$ is an even cycle $v_1, v_2, \ldots, v_{2k}, v_1$ (i.e., $\{v_i, v_{i+1}\} \in E(G^c)$, $i = 1, \ldots, 2k$, and $\{v_{2k}, v_1\} \in E(G^c)$), then $V(H)$ defines two disjoint cycles $C_H = C1_H \cup C2_H$ that are connected components in AG: $v_1, v_3, \ldots, v_{2k-1}, v_1$ and $v_2, v_4, \ldots, v_{2k}, v_2$.

Lemma 3. *Let c^t such that $sum(c^t) = (n+1)/2$. Let H be a connected component of G^c on l vertices, $l \geq 2$. Let $S = \{v \in H : c_v^t = 1\}$. Then*

$$|\{v \in H : c_v^{t+1} = 1\}| \geq l - |S| \tag{4}$$

Furthermore, the equality holds in (4) if and only if one of the following holds: (i) $|S| = 0$, (ii) $|S| = l$, (iii) H is an even cycle and $c_v^t \neq c_w^t$ whenever $\{v, w\} \in E(G^c)$.

Proof. First note that, by Lemma 2 and by definition of AG,

$$|\{v \in H : c_v^{t+1} = 1\}| = \sum_{v \in H} c_v^{t+1} = \sum_{\{u,w\} \in C_H \subset AG} (1 - c_u^t c_w^t) = |H| - \sum_{\{u,w\} \in C_H \subset AG} c_u^t c_w^t.$$

Thus, it remains to show that

$$|S| \geq \sum_{\{u,w\} \in C_H \subset AG} c_u^t c_w^t. \tag{5}$$

Note that

$$\sum_{\{u,w\} \in C_H \subset AG} c_u^t c_w^t = |\{\{u,w\} \in E_{AG}(C_H) : u, w \in S\}| = |E(C_H[S])|$$

where $C_H[S]$ denotes the induced subgraph of C_H, i.e., the maximal subgraph of C_H on the vertex set $S \subset V(C_H)$. If $|S| = 0$, $|E(C_H[S])| = 0$, and (5) holds with equality. Thus, (4) holds with equality. If $|S| = l$, then $C_H[S] = C_H$ and $|E(C_H[S])| = |E(C_H)| = l$ since C_H is a cycle or a union of two disjoint cycles. Thus, if $|S| = l$, (4) also holds with equality. If H is an even cycle, then $C_H = C1_H \cup C2_H$. Furthermore, $S = V(C1_H)$ or $S = V(C2_H)$ if and only if

vertices of H are colored alternately along the cycle H (i.e., as described in (iii) in the statement of the lemma). In either case, $|E(C_H[S])| = |S|$ and (4) again holds with equality. If neither (i) nor (ii) nor (iii) holds, then $C_H[S]$ contains an acyclic component and any possible cyclic component of C_H must be a cycle.[5] Thus, $|E(C_H[S])| \leq |S| - 1$ and $|\{v \in H : c_v^{t+1} = 1\}| \geq l - |S| + 1$. □

Several simple consequences of this lemma will be useful in the analysis that follows. For example, if a connected component of G^c that is not an isolated vertex is monochromatic for some c^t, then every vertex in H will switch color.

Lemma 4. *Let c^t be a coloring of G, $\delta(G) \geq n - 3$, such that $sum(c^t) = (n + 1)/2$. Let $H = (V_H, E_H)$ be a connected component of G^c with $|V_H| \geq 2$. Suppose that $c_v^t = c_w^t$ for every $v, w \in V_H$. Then $c_v^{t+1} = 1 - c_v^t$ for every $v \in V_H$.*

The next lemma presents an opposite scenario: if colors assigned by c^t, $sum(c^t) = (n + 1)/2$, alternate along an even cycle that is a connected component of G^c, then no vertex on that cycle will switch color.

Lemma 5. *Let c^t be a coloring of G, $\delta(G) \geq n - 3$, such that $sum(c^t) = (n + 1)/2$. Let $C_{2k} \subset G^c$ be a connected component in G^c. Suppose the colors assigned by C^t alternate along the cycle: if u is adjacent to v in C_{2k} then $c_u^t = 1 - c_v^t$. Then $c_v^{t+1} = c_v^t$ for every $v \in C_{2k}$.*

The preceding lemmas indicate a way to construct c^0 yielding a complete switch, i.e., $c^1 = 1 - c^0$. Obviously, all masters must be colored with a minority color in order to switch. If all the other connected components of G^c are monochromatic (with some even cycles possibly being colored as described in the previous lemma), and if the resulting coloring c^0 is a tight majority coloring on G (i.e., $sum(c^t) = (n+1)/2$), then, as shown in the next lemma, $c^1 = 1 - c^0$ (except on the even cycles where $c^{t=1} = c^t$), and G is not a majority consensus computer.

Lemma 6. *Let $\delta(G) \geq n - 3$. Let $H_1 = (V_1, E_1^c)$, $H_2 = (V_2, E_2^c)$, ..., $H_m = (V_m, E_m^c)$ be the connected components of G^c. Suppose there exist i and j, $1 \leq i < j \leq m$, such that*
(i) $|V_k| = 1 \Rightarrow k \leq i$,
(ii) $m \geq k > j \Rightarrow H_k$ is an even cycle,
(iii) $|V_1| + |V_2| + \ldots + |V_i| + 1 = |V_{i+1}| + \ldots + |V_j|$.
Then G is not a majority consensus computer.

Proof. For $v \in V_k$, set $c_v^0 = 0$ if $k \leq i$ and set $c_v^0 = 1$ if $i < k \leq j$. If $j < m$, then the remaining vertices lie on even cycles in G^c. Color each H_k alternately, i.e., as described in the statement of Lemma 5. Note that, by (iii),

$$\begin{aligned} |\{v \in V : c_v^0 = 0\}| &= \textstyle\sum_{k=1}^{i} |V_k| + \frac{1}{2} \sum_{k=j+1}^{m} |V_k| \\ &= (\textstyle\sum_{k=i+1}^{j} |V_k|) - 1 + \frac{1}{2} \sum_{k=j+1}^{m} |V_k| \\ &= |\{v \in V : c_v^0 = 1\}| - 1. \end{aligned}$$

Thus, $sum(c^0) = (n + 1)/2$ and $maj(c^0) = 1$.

[5] In fact the only possibility for a cyclic component is when H is an even cycle

If v is a master $c_v^1 = maj(c^0) = 1 = 1 - c_t^0$ (the last equality holds because $\{v\} = H_k$ for some k and $k \leq i$ by (i)). If v is not a master, then $v \in H_k$ for some $k \leq m$ such that $|H_k| \geq 2$. If $k \leq j$, then $c_v^1 = 1 - c_v^0$ by Lemma 4. If $k > j$ then $c_v^1 = c_v^0$ by Lemma 5. Therefore, $c_v^1 = 1 - c_v^0$ if $v \in V_1 \cup \ldots \cup V_j$ and $c_v^1 = c_v^0$ if $v \in V_{j+1} \cup \ldots \cup V_m$. Thus,

$$\begin{aligned} |\{v \in V : c_v^1 = 1\}| &= \textstyle\sum_{k=1}^{i} |V_k| + \frac{1}{2} \sum_{k=j+1}^{m} |V_k| \\ &= (\textstyle\sum_{k=i+1}^{j} |V_k|) - 1 + \frac{1}{2} \sum_{k=j+1}^{m} |V_k| \\ &= |\{v \in V : c_v^1 = 0\}| - 1. \end{aligned}$$

So, $maj(c^1) = 0 \neq maj(c^0)$ and G is not a majority consensus computer by Proposition 1. □

For any $k = 0, 1, \ldots (n-1)/2$, it is straightforward to construct a G with k masters satisfying conditions of Lemma 6. For example, if $k = 0$, take G such that connected components of G^c are $P_{(n-1)/2}$ and $P_{(n+1)/2}$. If $k > 0$, G whose connected components are its masters, P_{k+1} and C_{n-2k-1} is such example. Thus, there exist G with $\delta(G) \geq n-3$ which are not m.c.c. and having exactly k masters for every $k < (n+1)/2$. (Recall that, by Proposition 3, every G with at least $(n+1)/2$ masters is a m.c.c.)

In order to prove that the Master Conjecture holds in the case $\delta(G) \geq n-3$, we need yet another lemma. In what follows we will say that $v_1, v_2, \ldots v_k$ form a path P_k if v_i is adjacent to v_{i+1} in P_k for $i = 1, \ldots, (k-1)$. Similarly, we will say that $v_1, \ldots v_k$ form a cycle C_k if $v_1, \ldots, v_k$ form a path $P_k \subseteq C_k$ and v_1 is adjacent to v_k in C_k.

Lemma 7. *Let c^t be a coloring of G, $\delta(G) \geq n-3$, such that $sum(c^t) = (n+1)/2$. Let $v_1, v_2, \ldots, v_k$ form $H \subset G^c$, a connected component in G^c on $k \geq 3$ vertices. Suppose that there exists a $j < k/2$ such that $c_{v_i}^t = i \bmod 2$ for $i \leq 2j+1$. If $2j+1 < k$, also suppose that $c_{v_i}^t = c_{v_{2j+2}}^t$ for $i > 2j+1$.*

Then $c_{v_i}^{t+1} = c_{v_i}^t$ for $i \leq 2j+1$ and $c_{v_i}^{t+1} = 1 - c_{v_i}^t$ for $i > 2j+1$.

Proof. Since $\delta(G) \geq n-3$, H is a path or a cycle. Using (b) and (c) of Lemma 2, observe that $c_{v_i}^{t+1} = c_{v_i}^t$ for $i \leq 2j+1$ (since each v_i such that $c_{v_i}^t = 0$ has both non-neighbors of color 1, while each v_i such that $c_{v_i}^t = 1$ has at least one non-neighbor of color 0) and that $c_{v_i}^{t+1} = 1 - c_{v_i}^t$ for $i > 2j+1$ (if $c_{v_{2j+2}}^t = \ldots = c_{v_k}^t = 0$, then each such v_i has a non-neighbor of color 0; if $c_{v_{2j+2}}^t = \ldots = c_{v_k}^t = 1$, then each such v_i has all non-neighbors of color 1 because $c_{v_1}^t = c_{v_{2j+1}}^t = 1$). □

Theorem 5. *Let G such that $\delta(G) \geq n-3$. If G is a majority consensus computer, then G contains a master.*

Proof. Suppose G does not contain a master. We'll show that G is not a majority consensus computer. Let $H_1 = (V_1, E_1^c)$, $H_2 = (V_2, E_2^c)$, ..., $H_m = (V_m, E_m^c)$ be the connected components of G^c. Since G does not contain a master, $|V_l| \geq 2$, $l = 1, \ldots m$. Choose an index i such that

$$|V_1| + \ldots + |V_i| \leq (n-1)/2 < |V_1| + \ldots + |V_i| + |V_{i+1}|.$$

If $|V_1| + \ldots + |V_i| = (n-1)/2$, then the conditions of Lemma 6 are satisfied with i, and with $j = m$. Therefore, in this case, G is not a majority consensus computer.

For the rest of the proof we may assume that $|V_1| + \ldots + |V_i| < (n-1)/2$. We may also assume that $|V_1| + \ldots + |V_{i-1}| + |V_i| + (|V_{i+1}|/2) > (n-1)/2$. (If not, then $(|V_{i+1}|/2) + |V_{i+2}| + |V_{i+3}| + \ldots + (|V_m|) > (n-1)/2$ and we could map l to $m+1-l$, i.e. H_l becomes H_{m+1-l}, $l = 1, \ldots, m$.) Note that these imply that $|V_i| \geq 3$.

Let $v_1, v_2, \ldots, v_k$ form H_{i+1} and let

$$j = (n-1)/2 - (|V_1| + \ldots + |V_{i-1}| + |V_i|) \tag{6}$$

Note that $j < k/2$. Set

$$c_v^0 = \begin{cases} 0 & v \in V_1 \cup V_2 \cup \ldots \cup V_i \\ i \mod 2 & v_i, \quad i = 1, \ldots, 2j+1 \\ 1 & v_i, \quad i = 2j+2, \ldots, k \\ 1 & v \in V_{i+2} \cup V_{i+3} \cup \ldots \cup V_m \end{cases}$$

Note that $sum(c^0) = (n+1)/2$. By Lemma 4, $c_v^1 = 1 - c_v$ for every $v \notin V_{i+1}$. By Lemma 7, $c_{v_i}^1 = 1 - c_{v_i}^0$ for $i = 2j+2, \ldots, k$ and $c_{v_i}^1 = c_{v_i}^0$ for $i = 1, \ldots 2j+1$. Thus, only j vertices colored by 0 and only $j+1$ vertices colored by 1 do not switch color. Hence, $sum(c^1) = |V_1| + \ldots + |V_i| + (j+1) = (n-1)/2 + 1 = (n+1)/2$ (the second equality follows from (6)).

Repeating the same argument for

$$c_v^1 = \begin{cases} 1 & v \in V_1 \cup V_2 \cup \ldots \cup V_i \\ i \mod 2 & v_i, \quad i = 1, \ldots, 2j+1 \\ 0 & v_i, \quad i = 2j+2, \ldots, k \\ 0 & v \in V_{i+2} \cup V_{i+3} \cup \ldots \cup V_m \end{cases}$$

we conclude that $c^2 = c^0$. Thus, $c^0, c^1, c^2, \ldots$ has period two. Therefore, G is not a majority consensus computer. □

5 Generalizations and Conclusions

The main result of this paper is that failure-free computation of majority consensus by iterative applications of the local majority rule is possible only in the networks that are nowhere truly local (Theorem 4). In other words, the idea of solving a truly global task (reaching consensus on majority) by means of truly local computation only (local majority rule) is doomed for failure. However, even well connected networks of agents that are nowhere truly local might fail to reach majority consensus when iteratively applying the local majority rule. We have investigated the properties of majority consensus computers, i.e., the networks in which iterative application of the local majority rule always yields consensus in the initial majority state.

There are several generalizations and relaxations of the local majority process that one might consider more realistic and applicable than the process we study.

Our results readily extend to many such generalizations and relaxations. For example, an obvious generalization of the model would be to allow weights on the edges and conduct the weighted local majority vote at each vertex at each time step according to those weights. If such weights on $E(G)$ are nonnegative real numbers, it is easy to see that there exists a multi-graph $M(G)$ on which our original process LMP mimics this weighted generalization of LMP on G. Here we only show one further result[6] that emphasizes our point that understanding LMP is fundamental to understanding any generalization of this process.

A simple generalization of the local majority process would allow vertex v to have some resistivity towards color switch. Formally, for a nonnegative integer k_v, we define a *k_v-local majority rule* for vertex v:

$$c_v^{t+1} = \begin{cases} c_v^t & \text{if } |\{w \in N_v : c_w^t = c_v^t\}| \geq \frac{|N(v)|}{2} + k_v \\ 1 - c_v^t & \text{if } |\{w \in N_v : c_w^t \neq c_v^t\}| > \frac{|N(v)|}{2} + k_v \end{cases} \tag{7}$$

The value k_v is called the *resistivity value* of vertex v and we call the graph $G = (V, E)$ together with the set of vertex resistivities $\{k_v : v \in V\}$ a *varied-resistivity* graph. Similarly, the process defined by (7) is called the local majority process with resistivities. Note that the local majority process with resistivities where $k_v = 0$, $v \in V$, is exactly the local majority process.

Theorem 6. *Let $G(V, E)$ be a varied-resistivity graph, with the vertex set $V = \{v_1, \ldots, v_n\}$, and the corresponding resistities $R = \{k_{v_1}, \ldots, k_{v_n}\}$. The local majority process with resistivities on the varied-resistivity graph G can be simulated by the local majority process on some graph $G'(V', E')$.*

Apart of generalizations of the model, there are several directions that might be of potential interest. One such direction is to determine the complexity of the decision problem:

MCC. *Input is a finite graph G. Is G a majority consensus computer?* Clearly, MCC is in co-NP because of Theorem 3 and it is very likely that MCC is co-NP complete. In [MP00], we solve this question by giving a complete characterization for graphs with $\delta(G) \geq (n-3)$, and a polynomial time algorithm that decides the problem.

We conjecture that every majority consensus computer G contains a master, i.e., there exists $v \in V(G)$ such that $d(v) = |V(G)| - 1$ (see Master Conjecture in Section 3).We have proved that this conjecture holds for almost complete networks, i.e., networks that are in a way most natural candidates for a counterexample to the conjecture (Theorem 5). However, the Master Conjecture remains open.

References

[Ber99] E. Berger. Dynamic monopolies of constant size. Manuscript, http://xxx.lanl.gov/abs/math/9911125, 1999.

[6] See [MP00] for detailed discussion.

[FLL+99] F. Flocchini, E. Lodi, F. Luccio, L. Pagli, and N. Santoro. Monotone dynamos in tori. In *Proc. 6th International Colloqium on Structural Information and Communication Complexity*, pages 152–165, 1999.

[FLLS98] P. Flocchini, E. Lodi, F. Luccio, and N. Santoro. Irreversible dynamos in tori. In *European Conference on Parallel Processing*, pages 554–562, 1998.

[GM90] E. Goles and S. Martinez. *Neural and Automata Networks.* Kluwer, Norwell MA, 1990.

[GO80] E. Goles and J. Olivos. Periodic behavior of generalized threshold functions. *Discrete Mathematics*, 30:187–189, 1980.

[GO81] E. Goles and J. Olivos. Comportement p'eriodique des fonctions 'a seuil binaires et applications. *Discrete Applied Mathematics*, 3:93–105, 1981.

[Gol86] E. Goles. *Positive automata networks*, pages 101–112. Disordered Systems and Biological Organization. Springer-Verlag, 1986.

[Has98] Y. Hassin. Probabilistic local polling processes in graphs. M.Sc. Thesis, The Weizmann Institute, Rehovot, Israel, 1998.

[HP99] Y. Hassin and D. Peleg. Distributed probabilistic polling and applications to proportionate agreement. In *Proc. 26th International Colloqium on Automata, Languages, and Programming*, pages 402–411, 1999.

[HP00] Y. Hassin and D. Peleg. Extremal bounds for probabilistic polling in graphs. In *Proc. 7th International Colloqium on Structural Information and Communication Complexity*, 2000.

[LPS99] F. Luccio, L. Pagli, and H. Sanossian. Irreversible dynamos in butterflies. In *Proc. 6th International Colloqium on Structural Information and Communication Complexity*, pages 204–218, 1999.

[Mor94a] G. Moran. Parametrization for stationary patterns of the r-majority operators on 0–1 sequences. *Discrete Mathematics*, 132:175–195, 1994.

[Mor94b] G. Moran. The r-majority vote action on 0–1 sequences. *Discrete Mathematics*, 132:145–174, 1994.

[Mor95] G. Moran. On the period-two-property of the majority operator in infinite graphs. *Trans. Amer. Math. Soc.*, 347(5):1649–1667, 1995.

[MP00] N. H. Mustafa and A. Pekeč. Democratic consensus and the local majority rule. Research Series RS-00-08, BRICS, University of Aarhus, Denmark, 2000.

[NIY99] T. Nakata, H. Imahayashi, and M. Yamashita. Probabilistic local majority voting for the agreement problem on finite graphs. In *Proc. 5th Computing and Combinatorics Conference*, pages 330–338, 1999.

[NIY00] T. Nakata, H. Imahayashi, and M. Yamashita. A probabilistic local polling game on weighted directed graphs with an application to the distributed agreement problem. *Networks*, 35(4):266–273, 2000.

[Pel98] D. Peleg. Size bounds for dynamic monopolies. *Discrete Applied Mathematics*, 86:263–273, 1998.

[PS83] S. Poljak and M. Sura. On periodical behaviour in societies with symmetric influences. *Combinatorica*, 3(1):119–121, 1983.

[PT86a] S. Poljak and D. Turzik. On an application of convexity to discrete systems. *Discrete Applied Mathematics*, 13:27–32, 1986.

[PT86b] S. Poljak and D. Turzik. On pre-periods of discrete influence systems. *Discrete Applied Mathematics*, 13:33–39, 1986.

Solvability of Equations in Free Partially Commutative Groups Is Decidable

Volker Diekert[1] and Anca Muscholl[2]

[1] Institut für Informatik, Universität Stuttgart,
Breitwiesenstr. 20-22, D-70565 Stuttgart
diekert@informatik.uni-stuttgart.de
[2] LIAFA, Université Paris VII,
2, place Jussieu, case 7014, F-75251 Paris Cedex 05
muscholl@liafa.jussieu.fr

Abstract. Trace monoids are well-studied objects in computer science where they serve as a basic algebraic tool for analyzing concurrent systems. The question whether the existential theory of trace equations is decidable has been solved positively in 1996. Free partially commutative groups (graph groups) generalize trace monoids in the sense that every element has an inverse. In this paper we show that the existential theory of equations over graph groups is decidable, too. Our decision procedure is non-elementary, but if a certain graph theoretical parameter is viewed as a constant, then we obtain a PSPACE-completeness result. Restricting ourselves to trace monoids we still obtain a better complexity result, as it was known previously.

1 Introduction

Free partially commutative monoids (or *trace monoids*) serve as a basic algebraic tool for investigating concurrent systems, [15,16]. Atomic actions are represented by letters and independency of actions is reflected by an independence relation on the alphabet. The independency defines a partial commutation. If each atomic action a has an inverse $\overline{a}$, then, on the algebraic level, we switch from monoids to groups. This means that we work with free partially commutative groups, which in algebra are also known as *graph groups*, see e.g. [7].

We show that the existential theory of equations over graph groups is decidable. Our decision procedure is non-elementary, but if a certain graph theoretical parameter is viewed as a constant, then we can establish a PSPACE-completeness result. In fact, we generalize a recent result of [4], which is also the starting point for us.

In the simplest setting we ask whether a single word equation in unknowns and constants is solvable. This problem is well-known to be NP-hard. It becomes PSPACE-hard, as soon as we add regular constraints for the unknowns, simply because the intersection problem for regular languages is PSPACE-complete, [10]. Makanin proved the decidability of word equations [11] and Schulz extended this decidability result in order to include regular constraints [20]. Standard

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 543–554, 2001.

methods imply that the existential theory of word equations is decidable, once the problem of solving a single equation is settled. The same holds for equations over free groups and again Makanin proved decidability [12,13]. The scheme of Makanin in the case of free groups is however known to be non primitive-recursive, see [9]. Only when Plandowski invented a new method for solving word equations by some polynomial space bounded algorithm [19], the corresponding problem for free groups was reconsidered and Gutiérrez [8] succeeded in extending Plandowski's polynomial space algorithm to free groups. In fact, it has been possible to prove decidability (and PSPACE-completeness) of equations with rational constraints in free groups, see [4].

The situation in trace monoids turned out to be quite different and decidability for trace equations was shown only in 1996 by Matiyasevich, see [14,5]. It remained open whether the solvability of graph group equations is decidable, but it was known that rational constraints are too powerful: They lead to undecidability (although this is not treated here). The good notion turned out to be *normalized rational constraint* (this concept is introduced here). Moreover, the main result of [4] is not about free groups, but about *free monoids with involution*. This is another key: It is not necessary to reduce to the situation in free groups. We introduce the notion of trace monoid with involution and reduce stepwise to the case of free monoids with involution by removing independency. This reduction is the main technical contribution of our work. We are strictly more general than in the case of trace monoids, and when applying our method in this special case we still obtain a better complexity result as known previously.

2 Preliminaries

2.1 Free Partially Commutative Monoids with Involution

Let us recall some basic concepts of trace theory, for details we refer to [6]. By Γ we mean a finite alphabet which is equipped with an involution $^- : \Gamma \to \Gamma$. An involution is a mapping such that $\overline{\overline{a}} = a$ for all $a \in \Gamma$. By $I \subseteq \Gamma \times \Gamma$ we denote an *independence relation*, its complement is $D = (\Gamma \times \Gamma) \setminus I$. It is called a *dependence* relation and we demand both that D is reflexive and symmetric and that $(a, b) \in D$ implies $(\overline{a}, b) \in D$ for all $a, b \in \Gamma$. In particular, we have $(a, \overline{a}) \in D$ for all $a \in \Gamma$. The *free partially commutative* monoid $M(\Gamma, I)$ is defined by the quotient monoid $\Gamma^*/\{ab = ba \mid (a, b) \in I\}$. According to Mazurkiewicz [15] it is also called a *trace monoid* and its elements are *traces*. If the reference to (Γ, I) is clear, we also write $\mathbb{M}$ instead of $M(\Gamma, I)$. The length $|x|$ of a trace x is the length of any representing word. A letter $a \in \Gamma$ is called minimal (maximal resp.) in x, if we can write $x = ay$ ($x = ya$ resp.) for some $y \in \mathbb{M}$. The set of minimal (maximal resp.) elements consists of pairwise independent letters. The set of letters occurring in $x \in \Gamma^*$ or in $x \in \mathbb{M}$ is denoted $\mathrm{alph}(x)$. For $a \in \Gamma$ let $I(a) = \{b \in \Gamma \mid (a, b) \in I\}$. By 1 we denote the empty word, the empty trace, and the unit element in a group.

We shall use node-labeled directed acyclic graphs $[V, E, \lambda]$ in order to represent traces. We assume that V is finite and that $\lambda : V \to \Gamma$ is a labeling such

that $(\lambda(v), \lambda(v')) \in D$ implies either $(v, v') \in E^*$ or $(v', v) \in E^*$, so all dependent vertices are ordered. Then $[V, E, \lambda]$ defines a unique trace $x = [V, E, \lambda] \in \mathbb{M}$ in a canonical way. A trace $x \in \mathbb{M}$ may have many different representations. We say that $[V, E, \lambda]$ is a *dependence graph* of a trace $x \in \mathbb{M}$, if $[V, E, \lambda]$ represents x and in addition $(\lambda(v), \lambda(v')) \in D$ is equivalent to $(v, v') \in \mathrm{id}_\Gamma \cup E \cup E^{-1}$. Up to isomorphism, the dependence graph of x is unique, and so is its induced labeled partial order (pomset) $[V, E^*, \lambda]$ which is also denoted by $[V, \leq, \lambda]$.

The involution $^- : \Gamma \to \Gamma$ is extended to the free monoid Γ^* by $\overline{a_1 \cdots a_n} = \overline{a_n} \cdots \overline{a_1}$. Since the independence relation I is supposed to be compatible with the involution, the same definition transfers to traces. Hence $(\mathbb{M}, ^-)$ is a trace monoid with involution. Note that we have $\overline{[V, E, \lambda]} = [V, E^{-1}, \overline{\lambda}]$ and $\overline{[V, \leq, \lambda]} = [V, \geq, \overline{\lambda}]$ respectively, where $\overline{\lambda}(v) = \overline{\lambda(v)}$ for all $v \in V$.

2.2 Recognizable and Rational Subsets, Factor Traces

For a moment let M be any finitely generated monoid and $\psi : \Gamma^* \to M$ be a surjective homomorphism. A subset $L \subseteq M$ is called *recognizable*, if $\psi^{-1}(L)$ is a regular word language. Languages of the form $\psi(L)$ with $L \subseteq \Gamma^*$ being regular are called *rational*.

Henceforth by ψ we mean the canonical homomorphism from words to traces, $\psi : \Gamma^* \to \mathbb{M}$. Two words representing the same trace have the same length. Fixing some linear order on Γ we can choose for a given trace $x \in \mathbb{M}$ the lexicographical first word representing x, which is denoted by $\mu(x) \in \Gamma^*$. Clearly $\psi(\mu(x)) = x$. It is well-known [1] that $\mu(\mathbb{M}) \subseteq \Gamma^*$ is a regular word language. If $L \subseteq \mathbb{M}$ is recognizable, then $\psi^{-1}(L) \cap \mu(\mathbb{M})$ is regular, too; and L is a homomorphic image of some regular subset of $\mu(\mathbb{M})$. Ochmański's Theorem says that this correspondence is one-to-one between recognizable subsets of $\mathbb{M}$ and regular subsets of $\mu(\mathbb{M})$, see [17], [18, Thm. 6.3.12].

Let $x = [V, \leq, \lambda] \in \mathbb{M}$ be a trace where $\leq$ is the partial order induced by the dependence graph. A *factor* or *factor trace* is a trace $f \in \mathbb{M}$ such that we can write $x = pfq$. Given a factorization $x = pfq$, we find some $F \subseteq V$ such that the induced pomset of F in $[V, \leq, \lambda]$ represents f. Moreover, F has the property that whenever $v \leq v' \leq v''$ with $v, v'' \in F$, then $v' \in F$, too. Conversely, let $F \subseteq V$ be a subset with this property: $v \leq v' \leq v''$ with $v, v'' \in F$ implies $v' \in F$. Then we can factorize $x = pfq$ such that F represents the factor f, but due to partial commutation the factorization is not unique (even if $F \neq \emptyset$). There is another warning. Assume that $V = F \cup G$ is a disjoint union where F represents f and G represents g. This does not mean that x is a product of f and g, in general. Indeed, let $\{a, b\} \times \{c, d\} \subseteq I$ and $x = abcd$. Then we have $abcd = cadb$, so both $f = bc$ and $g = ad$ are factors. But x is not a product of f and g as soon as both $ab \neq ba$ and $cd \neq dc$.

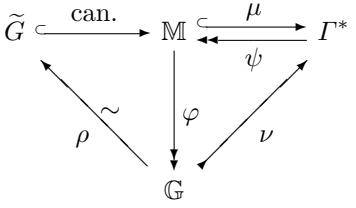
2.3 Graph Groups and Their Normalized Rational Subsets

We can switch from monoids with involution to groups by interpreting $\overline{a}$ as a^{-1}. Formally, we define the quotient monoid $G(\Gamma, I, ^-)$ by $M(\Gamma, I)/\{a\overline{a} = 1 \mid a \in \Gamma\}$,

and the canonical homomorphism is denoted by $\varphi : M(\Gamma, I) \to G(\Gamma, I, ^-)$. Since $\overline{\overline{a}} = a$, this is a group. It is called the *graph group* over $(\Gamma, I, ^-)$. This is a slight abuse of language, since the involution may have fixed points, i.e., we allow $a = \overline{a}$ for some $a \in \Gamma$. This freedom is very useful for various reasons. Here it means that $G(\Gamma, I, ^-)$ may have elements of order two. Torsion free groups arise if the involution has no fixed points. This is the usual setting of graph groups as investigated e.g. by Droms in [7]. If the reference to $(\Gamma, I, ^-)$ is clear we sometimes write $\mathbb{G}$ instead of $G(\Gamma, I, ^-)$. We can describe the group $\mathbb{G}$ by means of a confluent and Noetherian trace rewriting system $S = \{a\overline{a} \to 1 \mid a \in \Gamma\}$. Termination and confluence of S as a rewriting system over $\mathbb{M}$ can be easily verified. For a general treatment or trace rewriting systems the interested reader is referred to [3]. A trace $x \in \mathbb{M}$ is called *reduced*, if it is irreducible with respect to the system S, i.e., the trace x contains no factor of the form $a\overline{a}$ with $a \in \Gamma$. Then every group element $x \in \mathbb{G}$ has a unique reduced representation. This means there is a unique reduced trace $\rho(x) = \widetilde{x} \in \mathbb{M}$ such that $\varphi(\widetilde{x}) = x$.

The set of all reduced traces $\widetilde{G} \subseteq \mathbb{M}$ is therefore in one-to-one correspondence with $\mathbb{G}$ via some injective mapping ρ. Since $\widetilde{G}$ is defined by some finite set of forbidden factors, it is a recognizable subset of $\mathbb{M}$. Hence $\mu(\widetilde{G})$ is a regular language of words. The mapping $\nu : G(\Gamma, I, ^-) \to \Gamma^*$ which maps a group element x to $\nu(x) = \mu(\widetilde{x}) = \mu(\rho(x))$ is a normal-form mapping, i.e., $\varphi(\psi(\nu(x))) = x$. The word $\nu(x)$ is the lexicographical first word among the shortest words which represent x.

We obtain the following commutative diagram:



A group language $L \subseteq \mathbb{G}$ is called *normalized regular*, if the set of normal forms $\nu(L) \subseteq \Gamma^*$ is a regular word language. Since $\nu(\mathbb{G} \setminus L) = \nu(G) \setminus \nu(L)$, the class of normalized rational languages is an effective Boolean algebra. If $\mathbb{G}$ is a free group, then every rational language is normalized regular. This is non-trivial; it follows from [2]. In general, the class of normalized regular languages is strictly contained in the class of rational subsets.

3 Equations over Graph Groups and Trace Monoids with Involution

In the following Ω denotes a set of variables (or unknowns) and we let $^- : \Omega \to \Omega$ act as an involution on unknowns without fixed points. A *propositional formula* over Γ and Ω is inductively defined as follows. Atomic formulas are

either equations of the form $\alpha = \beta$, where $\alpha, \beta \in (\Gamma \cup \Omega)^*$, or constraints. A constraint is of the form $X \in L$, where $X \in \Omega$, and L is either recognizable (if we deal with monoids) or normalized regular (if we deal with graph groups). For the specification of L we may use some NFA (non-deterministic finite automaton) where the accepted language is either in $\mu(\mathbb{M})$ (if we deal with monoids) or in $\nu(\mathbb{G})$ (if we deal with graph groups). Propositional formulas are built from atomic formulas using the Boolean connectives. A formula in the existential theory is an existentially quantified sentence. An *assignment* for a propositional formula is a mapping $\sigma : \Omega \to \Gamma^*$ such that $\sigma(\overline{X}) = \overline{\sigma(X)}$ for all $X \in \Omega$. As usual, we extend σ to a homomorphism $\sigma : (\Gamma \cup \Omega)^* \to \Gamma^*$ by letting $\sigma(a) = a$ for $a \in \Gamma$. By extending σ further with the homomorphism $\Gamma^* \to \mathbb{M}$ (if we consider trace monoids) and with the homomorphism $\Gamma^* \to \mathbb{G}$ respectively (if we consider graph groups), we read σ also as $\sigma : (\Gamma \cup \Omega)^* \to \mathbb{M}$ and $\sigma : (\Gamma \cup \Omega)^* \to \mathbb{G}$. There will be no risk of confusion. The evaluation of a propositional formula under an assignment $\sigma : \Omega \to \Gamma^*$ is defined in a canonical way: The evaluation under σ of a constraint $X \in L$ is the truth value of $\mu(\sigma(x)) \in \mu(L)$ and of $\nu(\sigma(x)) \in \nu(L)$, respectively. The evaluation of a formula depends on whether we work with a trace monoid $\mathbb{M}$ or with a graph group $\mathbb{G}$. An existentially quantified sentence is true, if and only if the underlying propositional formula evaluates to *true* by some assignment.

Our main theorem states that the existential theory of trace monoids with involution is decidable. The group case becomes a corollary. This is due to the following lemma and proposition which are close to similar statements in [4].

Lemma 1. *Let $\widetilde{x}, \widetilde{y}, \widetilde{z} \in \mathbb{M}$ be reduced traces representing the group elements $x, y, z \in \mathbb{G}$. Then we have $xy = z$ in $\mathbb{G}$ if and only if there are traces $p, q, r \in \mathbb{M}$ such that $\widetilde{x} = pq, \widetilde{y} = \overline{q}r$, and $\widetilde{z} = pr$.*

Proposition 1. *The existential theory of equations with normalized regular constraints over $G(\Gamma, I, {}^-)$ is* PSPACE*-hard. There is a polynomial time reduction of this theory to the existential theory of equations with recognizable constraints over trace monoids with involution $M(\Gamma, I)$.*

4 The Main Theorem

In order to have a convenient complexity bound below we introduce the graph theoretical concept of a complete clan. A *complete clan* in (Γ, D) is a maximal subset $A \subseteq \Gamma$ such that $(a, c) \in D \Leftrightarrow (b, c) \in D$ for all $a, b \in A$, $c \in \Gamma$. Note that a complete clan is a complete subgraph of (Γ, D), since D is reflexive. The set of complete clans is a partition of Γ and we denote by $c(\Gamma, D)$ the number of complete clans. A complete clan A is called *thin*, if there are $a \in A$, $b \in \Gamma \setminus A$ such that $(a, b) \in I$. The following facts are easily verified: The number of thin clans is either $c(\Gamma, D)$ or $c(\Gamma, D) - 1$, it is never 1. If $\mathbb{M}$ is a direct product of d free monoids, then the number of thin clans is d for $d > 1$, and it is 0 for $d = 1$.

Recognizable constraints are a natural concept in the framework of trace equations: They are needed for negations and more importantly for expressing

commutation during the reduction. For this we introduce two macros: The formula

$$\bigvee_{a\in\Gamma}\big((X \in a\mathbb{M} \,\&\, Y \notin a\mathbb{M}) \vee (X \notin a\mathbb{M} \,\&\, Y \in a\mathbb{M})\big)$$

represents $\min(X) \neq \min(Y)$. With the help of this macro we can replace an inequality $\alpha \neq \beta$ by the equivalent formula $\exists Z \exists X \exists Y : \alpha = ZX \;\&\; \beta = ZY \;\&\; \min(X) \neq \min(Y)$.

A formula $\mathrm{alph}(X) \times \mathrm{alph}(Y) \subseteq I$ stands for

$$\bigwedge_{a\in\Gamma}(X \notin \Gamma^* a \Gamma^* \;\vee\; Y \in I(a)^*).$$

So, if $I = \emptyset$, then the macro means nothing but $X \in \{1\} \;\vee\; Y \in \{1\}$. A formula as above is called a *commutation constraint*.

Theorem 1. *Let $\tau \geq 0$. The following problem is* PSPACE*-complete.*

Input: *An existentially quantified sentence Φ of equations with recognizable constraints over a trace monoid with involution $(M(\Gamma, I), {}^-)$ such that the number of complete thin clans is at most τ.*

Question: *Is the sentence Φ true?*

Corollary 1. *Let $\tau \geq 0$. The following problem is* PSPACE*-complete.*

Input: *An existentially quantified sentence Φ of equations with normalized rational constraints over a graph group $G(\Gamma, I, {}^-)$ such that the number of complete thin clans is at most τ.*

Question: *Is the sentence Φ true?*

For $\tau = 0$ Thm. 1 follows by [4], since $\tau = 0$ is the situation of free monoids. We assume $\tau \geq 1$ and since τ is a constant, it is enough to give reduction to the case $\tau - 1$. This is done stepwise. First, we deal with negations: Using the first macro above (and the fact that the class of recognizable languages is closed under complementation) we may assume without restriction that Φ is a positive sentence. More precisely, for the polynomial time reduction we will assume that Φ may contain negations only on the constraints. We still call it a *positive sentence*.

Second, we make a reduction to the case where the set of fixed points $\Delta = \{a \in \Gamma \mid a = \overline{a}\}$ is empty. This reduction is omitted for lack of space. Hence, in the following we shall assume that the involution has no fixed points, i.e., $a \neq \overline{a}$ for all $a \in \Gamma$. Moreover, we shall assume that all equations of Φ are in triangulated form, $X = X_1 X_2$.

We fix for the rest of the paper some thin complete clan in (Γ, D) which we write in the form $A \cup \overline{A}$ with $A \cap \overline{A} = \emptyset$. We define $\widehat{D} = D \cup \Gamma \times (A \cup \overline{A}) \cup (A \cup \overline{A}) \times \Gamma$, so the clan is not thin anymore with respect to $\widehat{D}$. Let $\widehat{I} = \Gamma \times \Gamma \setminus \widehat{D}$, then $\widehat{\mathbb{M}} = M(\Gamma, \widehat{I})$ is a trace monoid with involution, but the number of thin complete clans in $(\Gamma, \widehat{D})$ is at most $\tau - 1$.

In the remaining of the section $\hat{\psi}$ means the canonical homomorphism $\hat{\psi} : \widehat{\mathbb{M}} \to \mathbb{M}$. By induction, it is enough to transform Φ into some equivalent sentence $\widehat{\Phi}$ over $\widehat{\mathbb{M}}$. The handling of recognizable constraints is trivial: A constraint $X \in L$ ($X \notin L$ resp.) is replaced by $X \in \hat{\psi}^{-1}(L)$ ($X \notin \hat{\psi}^{-1}(L)$ resp.).

For a replacement of an equation $X = X_1X_2$ let $d = 3\tau^2 + 8\tau + 7$ and choose d new, existentially quantified variables $Y_1, \ldots, Y_d$. The replacement is a disjunction over all permutations $\pi \in \mathrm{Perm}(d)$ and all c with $0 \leq c \leq d$. Each clause in the disjunction is the following conjunction of equations and commutation constraints:

$$X_1 = Y_1 \cdots Y_c \;\&\; X_2 = Y_{c+1} \cdots Y_d \;\&\; X = Y_{\pi(1)} \cdots Y_{\pi(d)}$$
$$\& \bigwedge_{(i-j)(\pi(i)-\pi(j))<0} \mathrm{alph}(Y_i) \times \mathrm{alph}(Y_j) \subseteq I$$

We obtain a system of equations with a larger set of variables and a new sentence $\widehat{\Phi}$. An easy reflection shows that if $\widehat{\Phi}$ evaluates to true over $\widehat{\mathbb{M}}$, then Φ evaluates to true over $\mathbb{M}$. Since τ is a constant, the size of $\widehat{\Phi}$ is polynomial in the size of Φ when using the commutation macro. In any case the transformation from Φ to $\widehat{\Phi}$ can be performed by some polynomially space bounded Turing machine. The heart of the paper is the proof of the converse: If Φ evaluates to true, then $\widehat{\Phi}$ evaluates to true as well.

5 Normal Forms

The aim of this section is to define a normal form mapping $\mathrm{nf} : \mathbb{M} \to \widehat{\mathbb{M}}$ which associates with each trace $x \in \mathbb{M}$ a unique trace $\mathrm{nf}(x) \in \widehat{\mathbb{M}}$ and which is compatible with the involution, i.e., we demand $\hat{\psi}(\mathrm{nf}(x)) = x$ and $\mathrm{nf}(\overline{x}) = \overline{\mathrm{nf}(x)}$. As we will see below the existence of the normal form relies on the following:

Lemma 2. *Let $a \in \Gamma$ such that $a \neq \overline{a}$ and let $w \in \{a, \overline{a}\}^*$ be any word. Then there exists a unique $k \geq 0$ such that $w \in a^*(\overline{a}a^*)^k(\overline{a}^*a)^k\overline{a}^*$.*
For the same k we also have $\overline{w} \in a^*(\overline{a}a^*)^k(\overline{a}^*a)^k\overline{a}^*$.

We view a trace x as a labeled pomset $[V, \leq, \lambda]$, $\lambda : V \to \Gamma$. We write $v \in x$ for a vertex $v \in V$ and $u \parallel v$ whenever u, v are incomparable with respect to $\leq$. We also write $(u, v) \in I$ if $(\lambda(u), \lambda(v)) \in I$, and analogously for $D = (\Gamma \times \Gamma) \setminus I$. Let $a_1 < \cdots < a_q$ be the linearly ordered subset of $(V, \leq)$ containing all vertices with label in the clan $A \cup \overline{A}$. We might have $q = 0$ meaning that there are no vertices with label in $A \cup \overline{A}$. We read $a_1 < \cdots < a_q$ as a word $a_1 \cdots a_q$ in the free monoid $(A \cup \overline{A})^*$. With each vertex $v \in V$ we shall associate the maximal factor of $a_1 \cdots a_q$ consisting of the vertices w with label in $A \cup \overline{A}$ which are incomparable with v, i.e., $w \parallel v$. For this, we introduce the *source* $s(v)$ and the *target point* $t(v)$ of v:

$$s(v) = \sup\{i \mid a_i \leq v\}, \quad t(v) = \inf\{i \mid v \leq a_i\},$$

where by convention, $\sup \emptyset = 0$ and $\inf \emptyset = q + 1$. Thus, $0 \leq s(v) \leq q$, $1 \leq t(v) \leq q + 1$ and $s(v) \leq t(v)$ for all $v \in V$. Note that we have $s(v) = t(v)$ if and only if the label of v belongs to $A \cup \overline{A}$.

For $0 \leq s \leq t \leq q + 1$ we define the *median position* $m(s, t)$. For $s = t$ let $m(s, t) = s$. For $s < t$ we choose by Lem. 2 the unique c with $s \leq c < t$ and

$k \geq 0$ such that $a_{s+1} \cdots a_c \in A^*(\overline{A}A^*)^k$ and $a_{c+1} \cdots a_{t-1} \in (\overline{A}^* A)^k \overline{A}^*$. Then we define $m(s,t) = c + \frac{1}{2}$ and we call $m(s,t)$ the *median position*. The median position $m(s(v), t(v))$ is called the *global position* of a vertex $v \in V$, it is denoted by $g(v)$. Hence $g(v) = m(s(v), t(v))$.

Lemma 3. *Let $x = [V, \leq, \lambda]$ and $v, w \in V$ be vertices such that $v \leq w$. Then we have $s(v) \leq s(w)$, $t(v) \leq t(w)$, and $g(v) \leq g(w)$.*

We define the normal form $\mathrm{nf}(x) \in \widehat{\mathbb{M}}$ by introducing *new arcs* into the dependence graph $[V, E, \lambda]$ of x. Let $v, w \in V$ such that $\lambda(w) \in A \cup \overline{A}$ and $v \parallel w$. (In particular, $\lambda(v) \notin A \cup \overline{A}$ and $g(v) \neq g(w)$.) We define a new arc from v to w, if $g(v) < g(w)$, otherwise we define a new arc from w to v. The arcs being already present in the dependence graph of x are called *old arcs*. The union $\hat{E}$ of old and new arcs defines a labeled directed graph $[V, \hat{E}, \lambda]$. From the lemma below it follows that $[V, \hat{E}, \lambda]$ is an acyclic graph:

Lemma 4. *Let $x \in \mathbb{M}$ and consider vertices u, v in x. For $u \leq v$ in $\mathrm{nf}(x)$ we have $g(u) \leq g(v)$. Moreover, we have $g(u) < g(v)$ whenever there is a path from u to v in $\mathrm{nf}(x)$ using at least one new arc.*

Thus, $[V, \hat{E}, \lambda]$ defines a unique trace $\mathrm{nf}(x) = [V, \hat{E}, \lambda]$ of $\widehat{\mathbb{M}}$. The important property of the normal form is $\mathrm{nf}(\overline{x}) = \overline{\mathrm{nf}(x)}$ for all $x \in \mathbb{M}$, which can be derived from Lem. 2.

Example 1. Let $x = a\,a\,\overline{a}\,\overline{a}\,a\,\overline{a}\,a\,\overline{a}\,\overline{a}\,a\,a\,b$ and $(a,b) \in I$. Then $k = 3$, $g(b) = 6\frac{1}{2}$ and $\mathrm{nf}(x) = a\,a\,\overline{a}\,\overline{a}\,a\,\overline{a}\,b\,a\,\overline{a}\,\overline{a}\,a\,a$.

Remark 1. A normal form satisfying $\mathrm{nf}(\overline{x}) = \overline{\mathrm{nf}(x)}$ for all $x \in \mathbb{M}$ cannot exist, in general, if the involution has fixed points. Indeed, assume we were in the situation $a, b \in A$, $a = \overline{a}$, $b = \overline{b}$, and $(a,b) \in I$. Then $\overline{ab} = ab$, but $(a,b) \in \widehat{D}$, so necessarily $\mathrm{nf}(\overline{ab}) \neq \overline{\mathrm{nf}(ab)}$ in $\widehat{\mathbb{M}}$.

After these preparations Thm. 1 becomes a consequence of the next one.

Theorem 2. *Let $d = 3\tau^2 + 8\tau + 7$ and let $x, x_1, x_2 \in \mathbb{M}$ such that $x = x_1 x_2$. Then there are $y_1, \ldots, y_d \in \widehat{\mathbb{M}}$, an index $0 \leq c \leq d$, and a permutation $\pi \in \mathrm{Perm}(d)$ with the following properties:*

$$\begin{aligned} \mathrm{nf}(x) &= y_{\pi(1)} \cdots y_{\pi(d)}, \\ \mathrm{nf}(x_1) &= y_1 \cdots y_c, \\ \mathrm{nf}(x_2) &= y_{c+1} \cdots y_d, \\ \mathrm{alph}(y_i) \times \mathrm{alph}(y_j) &\subseteq I \text{ for all } i, j, \text{ where } (i-j)(\pi(i) - \pi(j)) < 0. \end{aligned}$$

The proof of Thm. 2 covers the rest of the paper. First note that if the theorem is true for some $d \leq 3\tau^2 + 8\tau + 7$, then it is true for $d = 3\tau^2 + 8\tau + 7$. We let $x, x_1, x_2 \in \mathbb{M}$ such that $x = x_1 x_2$. We present x by its pomset $[V, \leq, \lambda]$ and $a_1 < \cdots < a_q$ is the linearly ordered subset of all vertices in V which have a label in $A \cup \overline{A}$. We define p to be the index such that $a_1, \ldots, a_p \in x_1$ and $a_{p+1}, \ldots, a_q \in x_2$. We have $0 \leq p \leq q$.

The case $q = 0$ is trivial (it could be excluded), the case $p = 0$ or $p = q$ is not trivial in the sense that we might have $\mathrm{nf}(x) \neq \mathrm{nf}(x_1)\,\mathrm{nf}(x_2)$, c.f. Ex. 1 above with $x_2 = b$.

In order to determine how $\mathrm{nf}(x)$ can be obtained from $\mathrm{nf}(x_1), \mathrm{nf}(x_2)$, we introduce the notion of *local position*. The local position $\ell(v)$ is the global position of v in x_1, if v belongs to x_1. If v belongs to x_2, then $\ell(v)$ is the global position of v in x_2 plus p, since we define $\ell(v)$ in x. Suppose that v is in x_1. With respect to x_1, the target point $t'(v)$ of v is $\min\{p+1, t(v)\}$, hence we define:

$$\ell(v) = m(s(v), \min\{p+1, t(v)\})\,.$$

Similarly, if v belongs to x_2 then the source point $s'(v)$ of v with respect to x_2 is $s'(v) = \max\{s(v), p\}$ and we define:

$$\ell(v) = m(\max\{s(v), p\}, t(v))\,.$$

The next proposition is a crucial technical result. It shows that among the vertices v with $s(v) \leq p < t(v)$, i.e., among the vertices where the local and global positions may differ, there is a constant number of different source points.

Proposition 2. *Let $x \in \mathbb{M}$ and define $S = \{s(v) \mid s(v) \leq p < t(v), v \in x\}$. Then we have $|S| \leq \tau + 1$.*

Proof. We may assume that $|S| \geq 2$. Choose a sequence $b_1, \ldots, b_k, b_{k+1} \in x$ with k minimal such that $S = \{s(b_i) \mid 1 \leq i \leq k+1\}$. We may assume that $0 \leq s(b_1) < \cdots < s(b_k) < p$ and $p < t(b_i)$ for $1 \leq i \leq k$. Hence we have $b_i \parallel a_p$ for all $1 \leq i \leq k$. We will show that $k \leq \tau$. For each $1 < i \leq k$ we choose a path from $a_{s(b_i)}$ to b_i in the dependence graph of x. On this path we pick a last vertex c_i with $c_i \leq a_p$. This vertex cannot be b_i. Hence there is a next vertex d_i with $(c_i, d_i) \in D$, $c_i < d_i \leq b_i$, and $d_i \parallel a_p$ for $1 < i \leq k$. Let $d_1 = b_1$. We claim that $\{d_i\} \times \{c_{i+1}, \ldots, c_k\} \subseteq I$ for $1 \leq i \leq k$. Indeed assume by contradiction that $(d_i, c_j) \in D$ for some $i < j \leq k$. Then $d_i \leq c_j$ since $s(b_i) < s(b_j)$. But we have $c_j \leq a_p$, hence $d_i \leq c_j$ implies $d_i \leq a_p$ in contradiction to $d_i \parallel a_p$.

Next, $(d_i, c_i) \in D$ and $\{d_i\} \times \{c_{i+1}, \ldots, c_k\} \subseteq I$ imply that c_i and c_j are in different thin clans for all $2 \leq i < j \leq k$. Moreover, there is another thin clan containing d_1. □

Analogously to Prop. 2 it holds that $|T| \leq \tau + 1$, where T is the set of target points: $T = \{t(v) \mid s(v) \leq p < t(v), v \in x\}$.

The positions we are interested in will be called in the following *cutting points*. The set of cutting points C is the union $C = C_g \cup C_\ell \cup \{0, p + \frac{1}{2}, q+1\}$ where

$$C_g = \{g(v) \mid s(v) \leq p < t(v), v \in x\},$$
$$C_\ell = \{\ell(v) \mid s(v) \leq p < t(v), v \in x\}.$$

Proposition 3. *The number of cutting points is bounded by $\tau^2 + 4\tau + 6$.*

The cutting points split the real interval $[0, q+1]$ into open intervals of the form (i,j) where $i,j \in C, i<j$, and $(i,j) \cap C = \emptyset$. There are $|C|-1$ such intervals, so the number is at most $\tau^2 + 4\tau + 5$. With each interval (i,j) we associate a factor trace of x. The factor trace is denoted either $x[0;i,j]$ or $x[3;i,j]$; we call $x[m;i,j]$, $0 \le m \le 3$, a *segment*. (Later on we will define segments with index 1 and 2.) Since $p+\frac{1}{2}$ is a cutting point we have either $j \le p+\frac{1}{2}$ or $p+\frac{1}{2} \le i$. For $j \le p+\frac{1}{2}$ we define

$$x[0;i,j] = \{v \in x \mid i < g(v) \le j \text{ and } t(v) \le p\}.$$

For $p+\frac{1}{2} \le i$ we define

$$x[3;i,j] = \{v \in x \mid i \le g(v) < j \text{ and } p < s(v)\}.$$

Note that we have $\ell(v) = g(v)$ for all $v \in x[0;i,j]$ or $v \in x[3;i,j]$. A segment as defined above is just a set of vertices of x. However, it is easy to see that a segment defines a factor trace of x and a factor trace of either x_1 or x_2. In the lemma below, we show that this property is still true for the normal forms $\mathrm{nf}(x), \mathrm{nf}(x_1), \mathrm{nf}(x_2)$.

Lemma 5. *Each segment $x[m;i,j], m = 0,3$, defines a factor trace y of $\mathrm{nf}(x)$. If $j \le p+\frac{1}{2}$, then $x[0;i,j]$ also defines a factor trace y_1 of $\mathrm{nf}(x_1)$ and we have $y = y_1$ in $\widehat{\mathbb{M}}$. If $p+\frac{1}{2} \le i$, then $x[3;i,j]$ also defines a factor trace y_2 of $\mathrm{nf}(x_2)$ and we have $y = y_2$ in $\widehat{\mathbb{M}}$.*

The segments $x[0;i,j], x[3;i,j]$ are pairwise disjoint subsets of x, but they do not cover x, in general. The missing points are those vertices v where $s(v) \le p < t(v)$. Therefore for each $m = 1,2$ and $i \le p < j$ we define an m-segment by:

$$x[m;i,j] = \{v \in x_m \mid s(v) = i \text{ and } t(v) = j\}\,, \quad m = 1,2\,.$$

Note that the indices i,j are not required to be cutting points. However, all vertices $v \in x[m;i,j]$ have the same local position $\ell(v)$ and the same global position $g(v)$, which are both cutting points. The number of non-empty 1 and 2 segments is at most $2(\tau+1)^2 = 2\tau^2 + 4\tau + 2$ by Prop. 2.

Lemma 6. *Each m-segment $x[m;i,j]$, $m = 1,2$ defines a factor trace y of $\mathrm{nf}(x)$ and a factor trace y_m of $\mathrm{nf}(x_m)$. We have $y = y_m$ in $\widehat{\mathbb{M}}$.*

Let us summarize the notations we have introduced up to this point. For each $m = 0,1,2,3$ we have defined m-segments of the form $x[m;i,j]$ which are pairwise disjoint subsets of x and cover x. Hence $F = \{x[m;i,j] \mid x[m;i,j] \neq \emptyset\}$ is a partition of the set x. Each $x[m;i,j]$ with $m \le 1$ is a factor trace of both $\mathrm{nf}(x)$ and $\mathrm{nf}(x_1)$ (respectively, each $x[m;i,j]$ with $m > 1$ is a factor trace of both $\mathrm{nf}(x)$ and $\mathrm{nf}(x_2)$). Using the counting above (Props. 2 and 3) we have $|F| \le 3\tau^2 + 8\tau + 7$.

By Lemma 5 and 6 we denote by $y_f \in \widehat{\mathbb{M}}$ the factor trace of $\mathrm{nf}(x)$ associated with $f \in F$. Since the segments in F cover all of $\mathrm{nf}(x), \mathrm{nf}(x_1)$ and $\mathrm{nf}(x_2)$ it remains to show how to write $\mathrm{nf}(x), \mathrm{nf}(x_1)$ and $\mathrm{nf}(x_2)$ as products of y_f, where f is in F. For this we have to compare segments. Every segment is associated

with either a pair of consecutive cutting points or a single cutting point. For a non-empty m-segment $f = x[m; i, j]$ we define a *global weight* $\omega_g(f)$ and a *local weight* $\omega_\ell(f)$ as follows. For $m \in \{0, 3\}$ let $\omega_g(f) = \omega_\ell(f) = \frac{i+j}{2}$, which is the center of the half-open interval associated with f. For $m \in \{1, 2\}$ let $\omega_g(f) = g(v)$ and $\omega_\ell(f) = \ell(v)$ for some $v \in f$. Recall that all $v \in f$ in this case have the same global position and the same local position.

We endow the set F of m-segments, $m \in \{0, 1, 2, 3\}$ with a global total order $\sqsubseteq_g$ and a local total order $\sqsubseteq_\ell$. In the following let $h \in \{g, \ell\}$, so h refers either to the global or to the local situation.

For $f = x[m; i, j]$, $f' = x[m'; i', j']$ we write $f \sqsubseteq_h f'$ if one of the following conditions holds:

1. $\omega_h(f) < \omega_h(f')$.
2. $\omega_h(f) = \omega_h(f')$, and $m < m'$.
3. $\omega_h(f) = \omega_h(f')$, $m = m'$, and $i < i'$.
4. $\omega_h(f) = \omega_h(f')$, $m = m'$, $i = i'$, and $j \leq j'$.

It is clear that $\sqsubseteq_g$ and $\sqsubseteq_\ell$ are both total orders. The next proposition has several important consequences. For example, it implies that $\sqsubseteq_g$ and $\sqsubseteq_\ell$ are both linearizations of the partial order $\leq$ of x.

Proposition 4. *Let $v, v' \in x$ be vertices such that $v \in f = x[m; i, j]$ and $v' \in f' = x[m'; i', j']$, where $f, f' \in F$. Then we have:*

- *If $v \leq v'$ or $g(v) < g(v')$ holds, then $f \sqsubseteq_g f'$.*
- *If $v \leq v'$ or $\ell(v) < \ell(v')$ holds, then $f \sqsubseteq_\ell f'$.*

Corollary 2. *Let $F = \{f_1, \ldots, f_d\}$ be sorted such that $f_i \sqsubseteq_g f_{i+1}$ for all i and let $y_i = y_{f_i}$ be the associated factors of $\mathrm{nf}(x)$. Then we have $\mathrm{nf}(x) = y_1 \cdots y_d$.*

Corollary 3. *Let $F = \{f_1, \ldots, f_d\}$ be sorted such that $f_i \sqsubseteq_\ell f_{i+1}$ for all i and let $y_i = y_{f_i}$ be the associated factors of $\mathrm{nf}(x)$. Then there exists some c with $0 \leq c \leq d$ satisfying*

$$\mathrm{nf}(x_1) = y_1 \cdots y_c \quad \textit{and} \quad \mathrm{nf}(x_2) = y_{c+1} \cdots y_d \,.$$

Corollary 4. *Let $f, f' \in F$ be segments such that $f \sqsubseteq_\ell f'$. Then $f' \sqsubseteq_g f$ and $f \neq f'$ imply $\mathrm{alph}(y_f) \times \mathrm{alph}(y_{f'}) \subseteq I$.*

For the final step we consider the normal forms of x, x_1, x_2 and express them as products of factor traces y_f associated with segments $f \in F$. We sort F such that $f_1 \sqsubseteq_\ell \cdots \sqsubseteq_\ell f_d$. Let $\pi \in \mathrm{Perm}(d)$ be the permutation such that $f_{\pi(1)} \sqsubseteq_g \cdots \sqsubseteq_g f_{\pi(d)}$ is F sorted with respect to $\sqsubseteq_g$. By Cor. 4 we have $\mathrm{alph}(y_i) \times \mathrm{alph}(y_j) \subseteq I$ whenever $(i - j)(\pi(i) - \pi(j)) < 0$. (As above, y_i is the factor trace associated with $f_i \in F$.) This completes the proof of Thms. 2 and 1.

Acknowledgment. We thank Yuri Matiyasevich for various contributions which were at the beginning of this work.

References

1. A. V. Anisimov and D. E. Knuth. Inhomogeneous sorting. *International Journal of Computer and Information Sciences*, 8:255–260, 1979.
2. M. Benois. Parties rationelles du groupe libre. *C. R. Acad. Sci. Paris, Sér. A*, 269:1188–1190, 1969.
3. V. Diekert. *Combinatorics on Traces.* LNCS 454. Springer, 1990.
4. V. Diekert, C. Gutiérrez, and C. Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE-complete. In *Proc. 18th Ann. Symp. on Theor. Aspects of Comp. Sci. (STACS'01)*, LNCS 2010:170–182, Springer, 2001.
5. V. Diekert, Yu. Matiyasevich, and A. Muscholl. Solving word equations modulo partial commutations. *Theoretical Computer Science*, 224:215–235, 1999. Special issue of LFCS'97.
6. V. Diekert and G. Rozenberg, editors. *The Book of Traces.* World Scientific, Singapore, 1995.
7. C. Droms. Isomorphisms of graph groups. *Proc. American Mathematical Society*, 100:407–408, 1987.
8. C. Gutiérrez. Satisfiability of equations in free groups is in PSPACE. In *Proc. 32nd Ann. ACM Symp. on Theory of Computing, STOC'2000*, pp. 21–27. ACM Press 2000.
9. A. Kościelski and L. Pacholski. Makanin's algorithm is not primitive recursive. *Theoretical Computer Science*, 191:145–156, 1998.
10. D. Kozen. Lower bounds for natural proof systems. In *Proc. 18th Ann. Symp. on Found. of Comp. Sci., FOCS'77*, pp. 254–266, IEEE Computer Society Press 1977.
11. G. S. Makanin. The problem of solvability of equations in a free semigroup. *Math. Sbornik*, 103:147–236, 1977. English transl. in Math. USSR Sbornik 32 (1977).
12. G. S. Makanin. Equations in a free group. *Izv. Akad. Nauk SSR*, Ser. Math. 46:1199–1273, 1983. English transl. in Math. USSR Izv. 21 (1983).
13. G. S. Makanin. Decidability of the universal and positive theories of a free group. *Izv. Akad. Nauk SSSR*, Ser. Mat. 48:735–749, 1984. In Russian; English translation in: *Math. USSR Izvestija, 25*, 75–88, 1985.
14. Yu. Matiyasevich. Some decision problems for traces. In *Proc. 4th Int. Symp. on Log. Found. of Comp. Sci. (LFCS'97)*, LNCS 1234: 248–257, Springer, 1997. Invited lecture.
15. A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.
16. A. Mazurkiewicz. Trace theory. In *Petri Nets, Applications and Relationship to other Models of Concurrency*, LNCS 255: 279–324, Springer, 1987.
17. E. Ochmański. Regular behaviour of concurrent systems. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 27:56–67, 1985.
18. E. Ochmański. Recognizable trace languages. In *The Book of Traces*, Chapter 6: 167–204. World Scientific, Singapore, 1995.
19. W. Plandowski. Satisfiability of word equations with constants is in PSPACE. In *Proc. 40th Ann. Symp. on Found. of Comp. Sci., FOCS'99*, pages 495–500. IEEE Computer Society Press 1999.
20. K. U. Schulz. Makanin's algorithm for word equations — Two improvements and a generalization. In *Word Equations and Related Topics*, LNCS 572: 85–150, Springer 1991.

Rational Transformations of Formal Power Series

Manfred Droste[1] and Guo-Qiang Zhang[2]**

[1] Institut für Algebra, Technische Universität Dresden
D-01062 Dresden, Germany
droste@math.tu-dresden.de
[2] Department of EECS, Case Western Reserve University
Cleveland, Ohio 44106, U.S.A.
gqz@eecs.cwru.edu

Abstract. Formal power series are an extension of formal languages. Recognizable formal power series can be captured by the so-called weighted finite automata, generalizing finite state machines. In this paper, motivated by codings of formal languages, we introduce and investigate two types of transformations for formal power series. We characterize when these transformations preserve rationality, generalizing the recent results of Zhang [15] to the formal power series setting. We show, for example, that the "square-root" operation, while preserving regularity for formal languages, preserves rationality for formal power series when the underlying semiring is commutative or locally finite, but not in general.

Keywords. Formal power series, rational languages, recognizable languages, weighted finite automata.

Introduction

In automata theory, Kleene's fundamental theorem on the equivalence of regular languages and finite automata has been extended in several ways. Schützenberger investigated formal power series over arbitrary semirings (such as the natural numbers) with non-commuting variables and showed that the recognizable formal power series [13], which represent precisely the behavior of automata with multiplicities (cf. Eilenberg [4]), coincide with the rational series. This was the starting point for a large amount of work on formal power series – see Kuich [9], Berstel-Reutenauer [1], Kuich-Salomaa [10], or Salomaa-Soittola [11] for surveys. Special cases of automata with multiplicities are networks with capacities (costs), which have been also investigated in operations research for algebraic optimization problems, cf. [16] and in the 'max-plus-community' [5].

Regular language operations such as union, concatenation, and star have their straightforward corresponding parts in formal power series. In fact, the concept of rational formal power series is based on these operations: *a formal*

** Corresponding author.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 555–566, 2001.

power series is rational precisely when it can be defined in terms of (a finite number of) these operations starting from the polynomials.

Many other operations on formal languages exist. An interesting and historically important class is captured by *regularity preserving functions* [7,12,14,15]. A simple yet nontrivial example is the so-called "square-root" operation: the square-root of a language L is the language $\mathsf{sqrt}(L) := \{w \mid ww \in L\}$, consisting of all the words w such that ww is in L. Although the square-root operation preserves regular languages, a closely related duplication operation, defined by $\{ww \mid w \in L\}$, clearly does not preserve regularity.

The square-root operation extends straightforwardly to formal power series. For a formal power series $S \in K\langle\langle A^* \rangle\rangle$, define $(\mathsf{sqrt}(S), w) := (S, ww)$, i.e., the coefficient of w in the new series is just the coefficient of ww in the original series. *Does* sqrt *preserve rational formal power series?* This is one of many questions that is easy to formulate, but nontrivial to answer. On the other hand, the notion of "rationality-preserving functions" takes some effort even to formulate precisely. In fact, it leads to two kinds of transformations with distinct properties.

The purpose of this paper is to introduce and investigate two types of transformations motivated from the coding of formal languages and their regularity-preserving properties. We provide characteristic conditions on the rationality-preserving property of these transformations, generalizing the recent results of Zhang [15] to the formal power series setting. Unlike the case for formal languages, the rationality-preserving property of these transformations critically depends on the property of the underlying semiring. The "square-root" operation, while preserving regularity for formal languages, preserves rationality for formal power series if the underlying semiring is commutative or locally finite, but not in general.

We note that for the Boolean semiring, there is already a rich class of regularity-preserving functions (see, e.g. [15]) which readily generalize to locally finite semirings by the results of this paper. These functions include polynomials and exponentials, and are preserved under most constructions such as sum, multiplication, exponentiation, but not subtraction. In fact, this function class is not even properly contained in the class of recursively enumerable functions, just to give an idea of how rich it is.

1 Formal Power Series and Weighted Finite Automata

We begin with the necessary notation and background for formal power series and for weighted finite automata. While a couple of introductory textbooks [10, 1] on formal power series are available, the concept of *weighted finite automata* remains folklore (or implicit in the literature). Weighted finite automata extract the computational content of recognizable formal power series. We recall the background here since this is a useful concept and several of our results depend on the intuitions provided by such a view.

A semiring is a structure $K = (K, +, \cdot, 0, 1)$ where $(K, +, 0)$ is a commutative monoid and $(K, \cdot, 1)$ is a monoid such that multiplication distributes

over addition, and $0 \cdot x = x \cdot 0 = 0$ for each $x \in K$. If the multiplication is commutative, we say that K is *commutative*. If the addition is idempotent, then the semiring is called *idempotent*. For instance, the Boolean semiring $\mathbb{B} = (\{0,1\}, +, \cdot, 0, 1)$ is both commutative and idempotent. The semiring of natural numbers $(\mathbb{N}, +, \cdot, 0, 1)$ is commutative but not idempotent.

Also, the semiring K is *locally finite* if any finitely generated subsemiring of K is finite. For instance, if both sum and product are commutative and idempotent then it is easy to see that the semiring is locally finite. This is in particular the case for the min-max semiring $(\mathbb{R} \cup \{-\infty, +\infty\}, \min, \max, +\infty, -\infty)$ or for semirings which are Boolean algebra, such as $(\mathcal{P}(A^*), \cup, \cap, \emptyset, A^*)$.

A formal power series is a mapping $S : A^* \to K$. It is usually denoted as a formal sum $S = \sum_{w \in A^*} (S, w)w$. The set $\mathrm{supp}(S) := \{w \in A^* \mid (S, w) \neq 0\}$ is called the *support* of S. If $\mathrm{supp}(S)$ is finite, then S is called a *polynomial*. The collection of all formal power series is denoted by $K\langle\langle A^* \rangle\rangle$. If $L \subseteq A^*$, we define the *characteristic series* of L by $1_L := \sum_{w \in L} 1w$. Let $S, T \in K\langle\langle A^* \rangle\rangle$ be two series. The *Hadamard product* $S \odot T \in K\langle\langle A^* \rangle\rangle$ is the series defined by

$$(S \odot T, w) := (S, w) \cdot (T, w) \qquad (w \in A^*).$$

It is known that if S, T are both recognizable and T is a characteristic series ($T = 1_L$ for some L), then $S \odot T$ is again recognizable, see [11], Theorem II.4.5.

In this paper we will be mostly concerned with formal power series where the underlying monoid is the freely-generated monoid A^* over a finite alphabet A. For this class of formal power series the machine concept of weighted finite automata is useful.

Definition 1. *A weighted finite automaton over a semiring $(K, +, \cdot, 0, 1)$ is a structure $W = (Q, A, \mu, \lambda, \gamma)$ where Q is a finite set of states, A is the input alphabet, $\mu : Q \times Q \times A \to K$ is the cost function, and $\lambda : Q \to K$ and $\gamma : Q \to K$ are cost functions for entering and leaving a state, respectively.*

The key distinction from the standard finite state machine is that in a weighted finite automaton, a cost is attached to a transition from one state to another while reading an input symbol. Weighted finite automata are inherently non-deterministic – the value 0 can be attached to impossible transitions. For simplicity, no ϵ-transitions are permitted.

The cost of a string $a_1 a_2 \cdots a_n$ along a path $p_1 \longrightarrow_{k_1}^{a_1} p_2 \longrightarrow_{k_2}^{a_2} p_3 \cdots \longrightarrow_{k_n}^{a_n} p_{n+1}$ in W is the product $\lambda(p_1)k_1k_2 \cdots k_n \cdot \gamma(p_{n+1})$. The cost of a string with respect to W is the sum of all the costs of the string along every distinct path. Thus, every weighted finite automaton determines a formal power series $K\langle\langle A^* \rangle\rangle$.

Rational formal power series are those which can be constructed using the operations of sum, product, and star (with the star operation applied only to those formal power series having 0 coefficient for the empty word). We write $K^{\mathrm{rat}}\langle\langle A^* \rangle\rangle$ for the set of rational formal power series over the semiring K.

Theorem 1 (Schützenberger). *A formal power series in $K\langle\langle A^* \rangle\rangle$ is rational if and only if it is the formal power series determined by some weighted finite automaton.*

Let $K^{n\times n}$ be the monoid of all $(n \times n)$-matrices over K, with matrix multiplication. A series $S \in K\langle\langle A^*\rangle\rangle$ is called *recognizable*, if there exists an integer $n \geq 1$, a monoid morphism $\mu : A^* \to K^{n\times n}$ and vectors $\lambda \in K^{1\times n}$, $\gamma \in K^{n\times 1}$ such that

$$(S, w) = \lambda \cdot \mu(w) \cdot \gamma$$

for each $w \in A^*$. We let $K^{\mathrm{rec}}\langle\langle A^*\rangle\rangle$ denote the collection of all recognizable formal power series $S \in K\langle\langle A^*\rangle\rangle$.

It can be seen that weighted finite automata correspond precisely to recognizable series: the cost function provides the generating matrices for the monoid morphism $\mu : A^* \to K^{n\times n}$. More intuitively, the cost for the automaton to go from state p to state q while reading an $a \in A$ is the (p, q)-entry in the matrix $\mu(a)$, assuming that states are labeled by consecutive integers starting from 1.

The generality of the semiring structure makes it possible to put many familiar examples in the context of weighted finite automata:

- $A = \{1\}$, $K = (\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$ where $\mathbb{R}_+$ is the set of non-negative real numbers. The weighted finite automata model shortest paths in the underlying graph.
- A, $K = ([0, 1], \max, \cdot, 0, 1)$. In this case the weighted finite automata model the probability/reliability of an action sequence, as in stochastic automata.
- $A = \{1\}$, $K = (\mathbb{R}_+ \cup \{\infty\}, \max, \min, 0, \infty)$. This models path capacity as in network flow analysis.
- A, Boolean semiring $\mathbb{B} = (\{0, 1\}, +, \cdot, 0, 1)$. This corresponds to classical non-deterministic finite automata theory.

We also note that Hashiguchi's solution to the restricted star-height problem [6] hinges upon a novel concept of the degree of non-determinism associated with a non-deterministic finite automaton, which can be formulated naturally as a weighted finite automaton. The power of weighted automata for the recognition of context-free languages was recently pointed out in [2].

2 Amplifying Transformations

Consider the following situation:

$$\begin{array}{ccc} A^* & \xrightarrow{\ h\ } & B^* \\ & & \Big\downarrow f \\ A^* & \xrightarrow{\ h\ } & B^* \end{array}$$

where h and f are *any* functions.

For any formal power series $S \in K\langle\langle A^*\rangle\rangle$, define, for every $w \in A^*$,

$$(\mathsf{A}(S, f, h), w) := \sum_{\substack{y\in A^* \\ h(y)=f(h(w))}} (S, y).$$

This is called an *amplifying transformation*. Intuitively, the entry/coefficient for w is the sum of the entries of all ys whose target under h is the same as the target of w under f composed with h.

The assumption is that h is a non-deleting epimorphism for most results of this section (non-deleting in the sense that for all w, $|\,w\,| \leq |\,h(w)\,|$), and length-preserving epimorphism for the next section. For convenience, B can sometimes be considered as a subset of A by renaming.

As an example, take $B = \{a\}$ to be a singleton, and take h to be the length function $w \longmapsto a^{|w|}$. The function f can be seen as a function from A^* to the natural numbers. For any language L (over the Boolean semiring), we have $w \in \mathsf{A}(L, f, h)$ (using the standard correspondence between formal languages and their characteristic series) if and only if there is a string $y \in L$ such that $|\,y\,| = f(|\,w\,|)$. In other words,

$$\mathsf{A}(L, f, h) = \{w \mid (\exists y \in L)\ |\,y\,| = f(|\,w\,|)\}.$$

This is precisely one of the language transformations considered in the literature [7,12,14,15].

It is both interesting and useful to note that such kind of transformations can be factored as the composition of familiar, more basic transformations on formal power series.

Let $h : A^* \to B^*$ be a function. If $T \in K\langle\langle B^* \rangle\rangle$, then

$$(h^{-1}T, w) := (T, h(w)) \qquad \text{for each } w \in A^*$$

defines a formal power series in $K\langle\langle A^* \rangle\rangle$. Similarly, for $S \in K\langle\langle A^* \rangle\rangle$,

$$(\bar{h}S, v) := \sum_{h(x)=v} (S, x) \qquad \text{for each } v \in B^*$$

defines a formal power series in $K\langle\langle B^* \rangle\rangle$.

For any homomorphism h, h^{-1} preserves rationality (see e.g. [10]). If h is a non-deleting homomorphism then $\bar{h}$ preserves rationality. If, moreover, h is length-preserving then one can obtain an explicit expression. Indeed, if $S \in K^{\mathrm{rec}}\langle\langle A^* \rangle\rangle$ is represented by (λ, μ, γ), then $\bar{h}S$ is given by $(\lambda, \bar{h}\mu, \gamma)$ where $\bar{h}\mu : B^* \to K^{n \times n}$ is the homomorphism defined by $(\bar{h}\mu)(v) := \sum_{x \in h^{-1}(v)} \mu x$.

The following lemma says that $\mathsf{A}(S, f, h)$ is nothing but the formal power series obtained by first transforming S in $K\langle\langle A^* \rangle\rangle$ to $\bar{h}S$ in $K\langle\langle B^* \rangle\rangle$ and then transforming $\bar{h}S$ under f^{-1} before being transformed back in $K\langle\langle A^* \rangle\rangle$ by h^{-1}.

Lemma 1 (Factorization Lemma). *For any function $h : A^* \to B^*$ and any $f : B^* \to B^*$, the following equality holds for any $S \in K\langle\langle A^* \rangle\rangle$:*

$$\mathsf{A}(S, f, h) = h^{-1} f^{-1} \bar{h} S.$$

Proof. For any $w \in A^*$, we have

$$(h^{-1} f^{-1} \bar{h} S, w) = (\bar{h}S, f(h(w))) = \sum_{\substack{y \in A^* \\ h(y) = f(h(w))}} (S, y) \ = (\mathsf{A}(S, f, h), w). \qquad \square$$

Let $h : A^* \to B^*$ be an epimorphism. For each $b \in B$ there exists a letter $a_b \in A$ such that $h(a_b) = b$. Let $X := \{a_b \mid b \in B\}$. Now define a linear order $\leq$ on A such that $x < y$ for each $x \in X$ and $y \in (A \setminus X)$. This order can be extended lexicographically to a linear order on A^*, also denoted by $\leq$. That is, for $v, w \in A^*$ put $v \leq w$ if either v is a prefix of w or there are $u \in A^*$ and $a, b \in A$ such that ua is a prefix of v, ub is a prefix of w, and $a < b$ in $(A, \leq)$.

Now for each word $w \in A^*$, the set $\{w' \in A^* \mid h(w') = h(w)\}$ is non-empty and contains, with respect to $\leq$, a smallest element $\breve{w}$ because $\leq$ is well-founded. This element $\breve{w}$ is called the *lexicographic normal form* of w (with respect to h). Let $\mathrm{LNF}(h)$ be the set of all lexicographic normal forms of words $w \in A^*$. One can show that $\mathrm{LNF}(h) = X^*$ and hence it is a rational language in A^* (we leave the proof for the full paper). Moreover, $|\mathrm{LNF}(h) \cap h^{-1}(v)| = 1$ for each $v \in B^*$.

Lemma 2. *Let $h : A^* \to B^*$ be an epimorphism, let $W \in K\langle\langle B^* \rangle\rangle$, and put $V = (h^{-1}W) \odot 1_{\mathrm{LNF}(h)}$. Then $\bar{h}V = W$, and if W is recognizable, then so is V.*

Proof. Since $\mathrm{LNF}(h)$ is a recognizable language in A^*, $1_{\mathrm{LNF}(h)}$ is a recognizable series (cf. [1]). Hence, if W is recognizable, so is V by the preceding remarks. Now, for each $w \in A^*$ we have

$$(\bar{h}V, w) = \sum_{\substack{x \in A^* \\ h(x) = w}} (W, h(x)) \cdot 1_{\mathrm{LNF}(h)}(x) \; = (W, w). \qquad \square$$

Here is our first main result of the section, which (together with the results of Section 4) appropriately generalizes the results of Seiferas and McNaughton (Theorem 5, [12]) and Kozen (Theorem 5, [8]) to the formal power series setting.

Theorem 2. *Suppose $h : A^* \to B^*$ is a non-deleting epimorphism and K a semiring. Given any function $f : B^* \to B^*$, the following are equivalent:*

1. *f^{-1} preserves recognizability (for series in $K\langle\langle B^* \rangle\rangle$).*
2. *the transformation $S \longmapsto \mathsf{A}(S, h, f)$ preserves recognizability (for series in $K\langle\langle A^* \rangle\rangle$).*

Proof. 1 $\Rightarrow$ 2: This follows directly from Lemma 1 and the fact that each of the transformations $\bar{h}$, f^{-1}, and h^{-1} preserves recognizability under the given conditions.

2 $\Rightarrow$ 1: Suppose for any $S \in K^{\mathrm{rec}}\langle\langle A^* \rangle\rangle$, $\mathsf{A}(S, h, f) \in K^{\mathrm{rec}}\langle\langle A^* \rangle\rangle$. Suppose also that $T \in K^{\mathrm{rec}}\langle\langle B^* \rangle\rangle$. By Lemma 2, $S := (h^{-1}T) \odot 1_{\mathrm{LNF}(h)}$ is a recognizable series in $K\langle\langle A^* \rangle\rangle$ and $\bar{h}S = T$. By assumption, $\mathsf{A}(S, h, f)$ is recognizable, and by Lemma 1 we have $\mathsf{A}(S, h, f) = h^{-1}f^{-1}T$. So, $\bar{h}((h^{-1}f^{-1}T) \odot 1_{\mathrm{LNF}(h)})$ is recognizable, and by Lemma 2 this series equals to $f^{-1}T$. $\square$

With exactly the same proof, we have the following result, substituting f^{-1} by $\bar{f}$ in the previous theorem.

Theorem 3. *Suppose $h : A^* \to B^*$ is a non-deleting epimorphism and K a semiring. Given any function $f : B^* \to B^*$, the following are equivalent:*

1. *$\bar{f}$ preserves recognizability (for series in $K\langle\langle B^* \rangle\rangle$).*
2. *the amplifying transformation $S \longmapsto h^{-1}\bar{f}\bar{h}S$ preserves recognizability.*

3 Coding Transformations

For coding transformations to make sense, we require $B \subseteq A$ for the homomorphism $h : A^* \to B^*$. But this is not a severe condition: in the case that h is a surjective homomorphism, one can always *rename* the letters in B to ensure $B \subseteq A$.

For any formal power series $S \in K\langle\langle A^* \rangle\rangle$, define, for every $v \in B^*$,

$$(\widetilde{h}S, v) := \sum_{\substack{x \in A^* \\ h(x)=v}} (S, vx).$$

Here the distinction from $\bar{h}$ is that the summation now is over (S, vx) instead of simply (S, x).

Note that if $A = B$ and $h : A^* \to A^*$ is the identity function, then $\widetilde{h}S = \mathsf{sqrt}(S)$. Next we show that if K is commutative, then any such transformation $\widetilde{h}$, hence in particular $\mathsf{sqrt}(S)$, preserves recognizability.

Proposition 1. *Let $h : A^* \to B^*$ be a length-preserving epimorphism, let K be commutative, and let $S \in K^{\mathrm{rec}}\langle\langle A^* \rangle\rangle$. Then $\widetilde{h}S$ is recognizable. In particular, $\mathsf{sqrt}(S)$ is recognizable.*

Proof. We first show the result for the sqrt-transformation, since this may be of independent interest. We then indicate how to modify the argument to get the general result for $\widetilde{h}S$.

Let $W = (Q, A, \mu, \lambda, \gamma)$ be a weighted finite automaton recognizing S. For each $i \in Q$, define $W'_i = (Q \times Q, A, \mu', \lambda', \gamma')$ by putting

- $\mu'((q, q'), (r, r'), a) = \mu(q, r, a) \cdot \mu(q', r', a)$,
- $\lambda'(q, q') = 0$ if $q' \neq i$; $\lambda'(q, i) = \lambda(q)$,
- $\gamma'(r, r') = 0$ if $r \neq i$; $\gamma'(i, r') = \gamma(r')$,

for any $q, q', r, r' \in Q$ and $a \in A$.

Intuitively, both components of W'_i are copies of W, with the first component having i as the only final state and the second component having i as the only initial state.

Now the executions of a word w in W'_i correspond bijectively to those executions of ww in W for which the first w leads to i (from which the second w continues), and since K is commutative, this correspondence preserves the costs. Let S'_i be the series recognized by W'_i. Then

$$(\mathsf{sqrt}(S), w) = (S, ww) = \sum_{i \in Q} (S'_i, w)$$

for each $w \in A^*$, showing that $\mathsf{sqrt}(S) = \sum_{i \in Q} S'_i$ is recognizable.

To obtain the result for $\widetilde{h}$, we replace in the above argument the second copy by an automaton for $\bar{h}S$, i.e., $\mu'((q, q'), (r, r'), b) = \mu(q, r, b) \cdot (\bar{h}\mu)(q', r', b)$, where $b \in B$, and we argue as before. □

Related to $\widetilde{h}$ is the *coding transformation*, defined as, for each $w \in A^*$,

$$(\mathsf{C}(S, f, h), w) := \sum_{\substack{y \in A^* \\ h(y) = f(h(w))}} (S, h(y)y).$$

Lemma 3. *For any $h : A^* \to B^*$ with $B \subseteq A$ and any function $f : B^* \to B^*$, the equality $\mathsf{C}(S, f, h) = h^{-1} f^{-1} \widetilde{h} S$ holds for any $S \in K\langle\langle A^* \rangle\rangle$.*

Proof. For any $w \in A^*$, we have

$$(h^{-1} f^{-1} \widetilde{h} S, w) = (\widetilde{h} S, f(h(w))) = \sum_{\substack{y \in A^* \\ h(y) = f(h(w))}} (S, h(y)y) = (\mathsf{C}(S, f, h), w).$$

□

We have a similar result to Proposition 1 when K is locally finite. The proof uses matrix-based techniques as given in Section 4.

Proposition 2. *If K is locally finite and h is a length-preserving epimorphism, then $\widetilde{h} S$ is rational for any rational S in $K\langle\langle A^* \rangle\rangle$.*

We also have the following useful result.

Proposition 3. *Suppose K is locally finite and $f : B^* \to B^*$ is any function. Then f^{-1} preserves recognizability of languages over B if and only if f^{-1} preserves recognizability of formal power series of $K\langle\langle B^* \rangle\rangle$.*

It is interesting to point out that the "only if" part is independent of the locally finite semiring K. The proof for this proposition uses some techniques developed in [3]; it is omitted here due to lack of space and will be included in the full paper.

Theorem 4. *Suppose $h : A^* \to B^*$ is a length-preserving epimorphism with $B \subseteq A$ and K is locally finite. Then for any function $f : B^* \to B^*$,*

1. *if f^{-1} preserves recognizability then the coding transformation preserves recognizability.*
2. *if B is a singleton, then the converse of (1) is also true.*

Proof. (1) Straightforward by Lemma 3 and Proposition 2.

(2) Choose any $S \in K^{\mathrm{rec}}\langle\langle A^* \rangle\rangle$ and consider the Hadamard product $S' = S \odot 1_{B^*}$. Then for each $w \in A^*$, we obtain

$$(\mathsf{C}(S', f, h), w) = \sum_{\substack{y \in B^* \\ h(y) = f(h(w))}} (S', h(y)y) = (S, f(h(w)) f(h(w)))$$

since h acts like the identity on B^* (i.e. $h(y) = y$ for $y \in B^*$). By Lemma 4 in the next section, the support of this series is a recognizable language in A^*. Putting $S = 1_L$, we obtain that for any recognizable language L in A^*, the language

$$L' = \{w \in A^* \mid f(h(w))^2 \in L\}$$

is recognizable. Hence

$$L' \cap B^* = \{w \in B^* \mid (f(w))^2 \in L\} = f^{-1}(\mathsf{sqrt}(L))$$

is recognizable in B^* for any recognizable language L. But since B is a singleton, any recognizable language in B^* is of the form $\mathsf{sqrt}(L)$ for some recognizable language $L \subseteq B^*$. This proves that f^{-1} preserves recognizability of languages and now we apply Proposition 3 to get the required conclusion. □

It should be interesting to compare this proof with the combinatorial proof of Kozen [8] for a related result for formal language transformations.

We end this section showing by a pumping argument that if K is not commutative, then in general even the sqrt-operation need not preserve recognizability.

Example. Let $K = \mathbb{B}\langle\langle\{a, b\}^*\rangle\rangle$, and let $S \in K\langle\langle\{a, b\}^*\rangle\rangle$ be given by $(S, w) = w$. Clearly, S is recognizable. We show that $T = \mathsf{sqrt}(S)$ is not recognizable. Indeed, suppose there was a weighted finite automaton W recognizing T. Say W has n states. Consider $w = ab^n$. Then $(T, w) = (S, ww) = w^2$. This cost equals the sum over the costs of all paths realizing w in W. But w^2 cannot be written as a proper sum in K. Due to the idempotence of K, the cost w^2 can be the sum of the costs of several paths realizing w; however, each of them has to have the same cost w^2. Fix a path realizing w with cost w^2. This path contains a loop which is labeled only with b's, say of length $j > 0$. So $w = ab^i b^j b^k$ where $i+j+k = n$, and the loop realizing b^j has some non-zero cost $c \in K$. Now consider $w_m = ab^i b^{mj} b^k = ab^{n+(m-1)j} (m \geq 2)$. Its cost in W is $ab^{n+(m-1)j} ab^{n+(m-1)j}$, which is obtained from the cost of the loop b^{mj}, so it has some power of c as a factor. If we choose m large enough, it follows that the fixed costs of the beginning sub-path labeled with ab^i and of the finishing sub-path labeled with b^k cannot contribute to the a in the middle of the cost of w_m. So the cost (word) c of the loop must contain an a. But then the cost of w_2 (containing the loop twice) would contain at least three a's, a contradiction.

We note that the above example would also work for the semiring $K = \mathbb{N}\langle\langle A^*\rangle\rangle$ with a similar argument.

4 Periodicity of Matrices and Recognizability-Preservation

For formal power series over a locally finite semiring K, the recognizability-preserving property can be characterized by the periodicity of matrices $K^{n\times n}$ – we establish results of this kind in this section.

Let $W = (Q, A, \mu, \lambda, \gamma)$ be a weighted finite automaton. We can take Q to be the initial segment $\{1, 2, 3, \ldots, n\}$ of positive integers and obtain a matrix Δ_a in $K^{n\times n}$ for each $a \in A$. For notational convenience, we write $\Delta(w)$ to be the matrix product $\Delta_{a_1}\Delta_{a_2}\cdots\Delta_{a_m}$, where $w = a_1a_2\cdots a_m$, and each a_i is a symbol in A.

We first have an observation (see [3] as well).

Lemma 4. *Let S be a series over a locally finite semiring K. If S is rational, then for each $k \in K$, the sliced series $S^{-1}(k)$, defined as $(S^{-1}(k), w) = 1$ if $(S, w) = k$ and $(S^{-1}(k), w) = 0$ otherwise, is again rational.*

Proof. Suppose $W = (Q, A, \mu, \lambda, \gamma)$ is a weighted finite automaton recognizing S with n states. Since μ is finite, we obtain a finite semiring $K' \subseteq K$ generated from the entries in μ. Construct a deterministic finite automaton $M = (P, A, \delta, q_0, F)$ as follows:

- $P = K'^{n\times n}$,
- $q_0 = I$ (the identity matrix as the starting state),
- $\delta(\Delta, a) = \Delta\,\mu(a)$ for each state Δ,
- Δ is a final state if and only if $\lambda\Delta\gamma = k$.

Then a string w is accepted by M precisely when $\lambda(\mu(w))\gamma = k$. □

Now consider the situation (where $a \in A$)

$$\begin{array}{ccc} A^* & \xrightarrow{\ h\ } & \{a\}^* \\ & & \Big\downarrow f \\ A^* & \xrightarrow{\ h\ } & \{a\}^* \end{array}$$

with h a non-deleting epimorphism. In this setting, we think of f as a function on natural numbers $\mathbb{N}$.

Definition 2. *Let K be a semiring. A function $f : \mathbb{N} \to \mathbb{N}$ is said to be ultimately periodic with respect to K-matrices if for each $n \geq 1$ and for each $\Delta \in K^{n\times n}$, there exists an integer $m > 0$ such that*

$$\Delta^{f(i)} = \Delta^{f(i+m)}$$

for all but finitely many $i \geq 0$.

Lemma 5. *Suppose $f : \mathbb{N} \to \mathbb{N}$ is ultimately periodic with respect to K-matrices. Then the amplifying transformation preserves recognizability.*

Proof. Let $W = (Q, A, \mu, \lambda, \gamma)$ be a weighted finite automaton that recognizes the formal power series $S \in K\langle\langle A^*\rangle\rangle$. We want to construct a weighted finite automaton $W' = (Q', A, \mu', \lambda', \gamma')$ recognizing $\mathsf{A}(S, f, h)$.

Since f is ultimately periodic with respect to matrices, for the matrix $\Delta := \sum_{a\in\Sigma}\mu(a)$ there exist integers $t, m > 0$ such that $\Delta^{f(j)} = \Delta^{f(j-m)}$ for all $j > t + m$. Now let W' be the following *deterministic* weighted finite automaton:

- $Q' = \{i \mid 0 \leq i \leq t+m\}$,
- $\lambda'(0) = 1$ and $\lambda'(i) = 0$ for $i > 0$ (so 0 is the only starting state),
- For each $a \in A$, $\mu'(i,j,a) = 1$ if either $j = i+1 \leq t+m$ or $i = t+m$ and $j = t+1$; otherwise $\mu'(i,j,a) = 0$,
- $\gamma'(i) = \lambda(\Delta^{f(i)})\gamma$.

The cost for a string $w \in A^*$ in W' is simply the exit cost $\lambda(\Delta^{f(|w|)})\gamma$ (by determinism), which is equal to $\sum_{|y|=f(|w|)}(S,y)$. □

The next result is concerned with the converse.

Lemma 6. *Suppose K is a locally finite semiring and suppose the amplifying transformation $\mathsf{A}(S,f,h)$ preserves recognizability for $S \in K\langle\langle\{a\}^*\rangle\rangle$. Then $f : \mathbb{N} \to \mathbb{N}$ is ultimately periodic with respect to K-matrices.*

Proof. For any $\Delta \in K^{n\times n}$, the series $S_{p,q}$ defined by $(S_{p,q}, a^i) = \Delta^i(p,q)$ is clearly recognizable, where $M(p,q)$ stands for the (p,q)-entry of the matrix M. Since K is locally finite, the semiring K' generated by entries in Δ is finite. By Lemma 4, for each $k \in K'$, $\{a^i \mid \Delta^i(p,q) = k\}$ is a regular language. Under the amplifying transformation, we see that $\{a^i \mid \Delta^{f(i)}(p,q) = k\}$ is again regular. This means that there is some $m > 0$ such that $\Delta^{f(i)}(p,q) = k$ if and only if $\Delta^{f(i+m)}(p,q) = k$ for all but finitely many $i \geq 0$. Since there are only finitely many choices for p, q, and $k \in K'$, we can obtain a common period $m' > 0$ such that $\Delta^{f(i)}(p,q) = k$ if and only if $\Delta^{f(i+m')}(p,q) = k$ for all p, q, all $k \in K'$, and all but finitely many $i \geq 0$. Therefore, $\Delta^{f(i)} = \Delta^{f(i+m')}$ for all but finitely many $i \geq 0$. □

The previous two lemmata and Theorem 2 lead to the following theorem.

Theorem 5. *Suppose K is a locally finite semiring. Then the following are equivalent:*

1. *$f : \mathbb{N} \to \mathbb{N}$ is ultimately periodic with respect to K-matrices.*
2. *f^{-1} preserves recognizability for $S \in K\langle\langle\{a\}^*\rangle\rangle$.*
3. *the amplifying transformation $\mathsf{A}(S,f,h)$ preserves recognizability for $S \in K\langle\langle\{a\}^*\rangle\rangle$.*

This result, together with the results of the previous sections, allows us to derive further results simply by chaining them together.

5 Conclusion

We have considered two kinds of transformations on formal power series and considered their recognizability-preserving properties. A rich class of functions have been shown to induce recognizability-preserving transformations by our characterization theorems.

Weighted finite automata provide a computationally intuitive representation of rational series. They serve as a valuable method for proving many results in this paper using the matrix-based idea described in [15].

Other variations of the transformations considered here are possible and those will be discussed in the full paper. We also note that although our results are formulated in the free monoid A^*, most of them generalize straightforwardly to general monoids.

References

1. J. Berstel and Ch. Reutenauer. *Rational Series and Their Languages*, volume 12 of *EATCS Monographs in Theoretical Computer Science*. Springer Verlag, 1988.
2. C. Cortes and M. Mohri. Context-free recognition with weighted automata, *Grammars* 3: 133–150, 2000.
3. M. Droste and P. Gastin. On aperiodic and star-free formal power series in partially commuting variables. In: *Formal power series and algebraic combinatorics (Moscow, 2000)*, 158–169, Springer, Berlin, 2000.
4. S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, New York, 1974.
5. S. Gaubert and M. Plus. Methods and applications of $(\max, +)$ linear algebra. In *Proceedings of STACS'97*, number 1200 in Lecture Notes in Computer Science, pages 261–282. Springer Verlag, 1997.
6. K. Hashiguchi. Algorithms for determining relative star height and star height. *Inform. and Comput.* 78 no. 2, 124–169, 1988.
7. S.R. Kosaraju. Regularity preserving functions. *SIGACT News*, 6 (2), 16–17, 1974.
8. D. Kozen. On regularity-preserving functions. Bulletin of the EATCS 58, 131-138, 1996.
9. W. Kuich. Semirings and formal power series: Their relevance to formal languages and automata. In G. Rozenberg and A. Salomaa, editors, *Handbook on Formal Languages*, volume 1, pages 609–677. Springer Verlag, 1997.
10. W. Kuich and A. Salomaa. *Semirings, Automata, Languages*, volume 6 of *EATCS Monographs in Theoretical Computer Science*. Springer Verlag, 1986.
11. A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Texts and Monographs in Computer Science. Springer Verlag, 1978.
12. J.I. Seiferas and R. McNaughton. Regularity preserving relations. *Theoretical Computer Science* 2:147–154, 1976.
13. M.P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
14. R.E. Stearns and J. Hartmanis. Regularity preserving modifications of regular expressions. *Information and Control*, 6 (1), 55–69, 1963.
15. G.-Q. Zhang. Automata, Boolean matrices, and ultimate periodicity. *Information and Computation* vol 152, 138–154, 1999.
16. U. Zimmermann. *Linear and Combinatorial Optimization in Ordered Algebraic Structures*, volume 10 of *Annals of Discrete Mathematics*. North Holland, 1981.

Combinatorics of Three-Interval Exchanges

Sébastien Ferenczi[1], Charles Holton[2], and Luca Q. Zamboni[3]

[1] Institut de Mathématiques de Luminy
CNRS - UPR 9016
Case 907, 163 av. de Luminy
F13288 Marseille Cedex 9 (France)
[2] Department of Mathematics
University of California
Berkeley, CA 94720-3840 (USA)
[3] Department of Mathematics
University of North Texas
Denton, TX 76203-5116 (USA)

Abstract. We generalize the interaction between Sturmian infinite words and rotations of the 1-dimensional torus by giving a set of necessary and sufficient conditions for a language to be a natural coding of a three-interval exchange. This solves an old question of Rauzy, and allows us to give a complete combinatorial description of such languages through an algorithm of simultaneous approximation.

1 Introduction

A well-known and well-working interaction between ergodic theory, arithmetics, and symbolic dynamics is provided by the study of irrational rotations on the torus $\mathbb{T}_1$. To an irrational real number α we associate a dynamical system, the rotation $Rx = x + \alpha \bmod 1$, an arithmetic algorithm, the usual continued fraction approximation, and a set of infinite words (a part of the class of Sturmian words) which are codings of trajectories under R by a canonical partition; the continued fraction algorithm forms the link between the dynamical system and the symbolic words, and the study of the arithmetic and symbolic objects is a great help for the study of the dynamical system. In this paper, we generalize this situation to the case of two real numbers $0 < \alpha < 1$ and $0 < \beta < 1$, starting from the dynamical system called a *three-interval exchange transformation* and defined on the interval $[0, 1[$ by

- $\mathcal{I}x = x + 1 - \alpha$ if $x \in [0, \alpha[$,
- $\mathcal{I}x = x + 1 - 2\alpha - \beta$ if $x \in [\alpha, \alpha + \beta[$,
- $\mathcal{I}x = x - \alpha - \beta$ if $x \in [\alpha + \beta, 1[$.

The *interval exchange transformations* have been introduced by Katok and Stepin [10]. They were further studied by Keane [11], Veech [16] and many others; it was Rauzy [14] who first saw the interval exchange transformations as a possible framework for generalizing the rotations/continued fractions/Sturmian

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 567–578, 2001.

words interaction, and the *Rauzy induction* is an algorithm of simultaneous approximation associated to interval exchange transformations. But the symbolic part was not complete, even in the simplest case of *three*-interval exchange transformations; indeed, several works ([6], [15]) were devoted to partial answers to the question asked by Rauzy "describe the category of infinite words which are natural codings of three-interval exchange transformations", a natural coding meaning a word (x_n) taking the value 1, 2, 3 when the n-th iterate of some point x lies in the first, second, third interval.

We are now able to answer the question of Rauzy, and give a full characterization of the infinite words, on three symbols, which are natural codings of three-interval exchange transformations with permutation (321); our main result is

Theorem 1. *A minimal infinite word is the natural coding of a non-trivial three-interval exchange transformation with* $2\alpha < 1$ *and* $2\alpha + \beta > 1$ *if and only if it satisfies the following four conditions:*

- *The factors of length* 2 *of* u *are* $\{12, 13, 21, 22, 31\}$.
- *If the* $w = w_1 \dots w_s$ *is a factor of* u*, its* retrograde $\bar{w} = w_s \dots w_1$ *is also a factor of* u.
- *For every* n *there are exactly two left special factors of length* n*, one beginning in* 1 *and one beginning in* 2.
- *If* w *is a bispecial factor ending in* 1 *and* $w \neq \bar{w}$ *then* $w2$ *is left special if and only if* $\bar{w}1$ *is left special.*

For all combinatorial definitions see section 2 below; "nontrivial" refers to a condition called i.d.o.c. described in subsection 3.1 below. If $2\alpha > 1$ or $2\alpha+\beta < 1$, we reduce this cases to the previous one by using *morphisms*, see subsection 4.2 below.

The "only if" part of our main result uses geometric considerations; the "if" part uses two powerful tools, the *negative slope expansion* which is the algorithm of simultaneous approximation defined and studied in [9], and the *hat algorithm* first defined in [13] for other types of infinite words: we start from an abstract language satisfying the four conditions of our theorem (we call such languages *symmetric* 3*-hat* because they are generated by the hat algorithm and linked to the geometry of three intervals), and the hat algorithm allows us to build it explicitely through its nested sequences of bispecial factors, this building being coded by a sequence $(m_k \in \mathbb{N}^*, n_k \in \mathbb{N}^*, \epsilon_{k+1} = \pm)_{k \geq 1}$. Then, when the symmetric 3-hat language does come from a three-interval exchange transformation, we are able to identify its coding sequence with the "partial quotients", for a variant of the negative slope expansion, of its parameters (α, β); thus we prove that every possible coding sequence can be associated to a three-interval exchange transformation, and this gives the "only if" part of our theorem.

And thus, as is the case with the link between Sturmian words and irrational rotations through the classical continued fraction approximation, this characterization gives also a full description of the language of three-interval exchange transformations for given (α, β), and, through a description of return words, an *S-adic presentation* of the dynamical system.

2 Combinatorics of Symmetric 3-Hat Languages

2.1 Symmetric Languages

Definition 2. Let A be a finite set. By a *language* L over A we mean a collection of sets $(L_n)_{n\geq 1}$ where each L_n consists of blocks of the form $a_1a_2\cdots a_n$ with $a_i \in A$ and such that for each $v \in L_n$ there exists $a, b \in A$ with $av, vb \in L_{n+1}$ and for all $v \in L_{n+1}$ if $v = au = u'b$ with $a, b \in A$ then $u, u' \in L_n$. We write $L = \cup_{n\geq 0} L_n \cup \{\varepsilon_\emptyset\}$ where $\varepsilon_\emptyset$ denotes the *empty word,* the unique word of length zero. The set A is called the *alphabet.*
The *complexity function* $p_L : \mathbb{N} \to \mathbb{N}$ is defined by $p_L(n) = \text{Card } (L_n)$.
A language L is *minimal* if for each $v \in L$ there exists n such that v is a factor of each word $w \in L_n$.
For an infinite word u, the language $L(u)$ is the set of its factors.

Definition 3. A word w in L is called *right special* (*left special*) if there exists distinct letters $a, b \in A$ such that both $wa, wb \in L$ ($aw, bw \in L$). If $w \in L$ is both right special and left special, then w is called *bispecial.* We regard $\varepsilon_\emptyset$ as bispecial.

Definition 4. If $u, w \in L$ we write

$$u \vdash w$$

if u is a prefix of w, and, for all $z \in L_{|w|}$ if u is a prefix of z then $z = w$;

$$u \models w$$

if w is the longest word so that $u \vdash w$.

In other words, $u \vdash w$ if w is the only extension of u with a given length. Thus if u is left special and $u \vdash w$ then w is also left special. If $u \vdash w$ and $u \vdash z$ then either $w \subseteq z$ or $z \subseteq w$. $u \models w$ if u is a prefix of only one word of length $|w|$ (namely w) and u is a prefix of at least two words of length $|w| + 1$. Thus if u is right special then $u \models u$. If u is left special and $u \models w$ then w is the shortest bispecial word beginning in u.

Lemma 5. *Let L be a minimal nonperiodic language. For all $u \in L$ there exists $w \in L$ so that $u \models w$. In other words for each u there exists a shortest right special word w beginning in u.*

Definition 6. Given a word $w = a_1a_2\cdots a_n$ with $a_i \in A$, let $\bar{w}$ denote the *retrograde word* of w, that is $\bar{w} = a_na_{n-1}\cdots a_1$. A word w is called a *palindrome* if $w = \bar{w}$.
We will call a language L *symmetric* if $w \in L$ implies $\bar{w} \in L$.

There are many examples of symmetric languages in the literature, the most well known being binary languages of complexity $p_L(n) = n+1$ called Sturmian languages [12] [4]. Other examples include languages derived from the Thue-Morse infinite word, Arnoux-Rauzy infinite words [1], and Episturmian infinite words [7].

Lemma 7. *Let L be a nonperiodic minimal symmetric language. Suppose $u \in L \setminus \{\varepsilon_\emptyset\}$, $a \in A$ and ua is left special.*

(1) If for all proper prefixes v of $\bar{u}$ the word av is not right special, then $ua \models ua\bar{u}$.
(2) Otherwise let v (possibly the empty word) be the longest proper prefix of $\bar{u}$ such that av is right special. Writing $\bar{u} = vx$ with $x \neq \varepsilon_\emptyset$ let b denote the first letter of x. Let w be the shortest word beginning in $\bar{v}a$ such that bw is right special. Writing $w = \bar{v}y$ then $ua \models uy$.

Lemma 8. *Let L be a nonperiodic minimal symmetric language with the property that for each left special word $u \in L \setminus \{\varepsilon_\emptyset\}$ there exists a unique letter $a \in A$ so that ua is left special. Let $u \in L \setminus \{\varepsilon_\emptyset\}$ be bispecial and $a \in A$ so that ua is left special. Suppose we are in case (2) of Lemma 7. Let v, x, b, w, and y be as in Lemma 7. Suppose further that v is non-empty.*

(1) If $\bar{v} = v$ then $a = b$, $w = \bar{u}$, $x = y$ and $\overline{uy} = uy$.
(2) If $\bar{v} \neq v$ then $w \neq \bar{u}$.

2.2 Symmetric 3-Hat Languages: Description of the Bispecial Words

Definition 9. A language L on the alphabet $\{1, 2, 3\}$ is called a *symmetric 3-hat language* if it satisfies the following conditions:

- $L_2 = \{12, 13, 21, 22, 31\}$.
- L is minimal and symmetric.
- For every n there are exactly two left special words of length n, one beginning in 1 and one beginning in 2.
- If w is a bispecial word ending in 1 and $w \neq \bar{w}$ then $w2$ is left special if and only if $\bar{w}1$ is left special.

Note that, because of the description of L_2, if w is left special, it has exactly two left extensions aw and bw; hence the third condition implies $p_L(n+1) - p_L(n) = 2$, hence $p_L(n) = 2n + 1$. We check that this third condition could be replaced by

- $p_L(n) = 2n + 1$.
- each bispecial word w is *o*rdinary ([3]), meaning that exactly three of the possible words awb are allowed.

Thus if $u \neq \varepsilon_\emptyset$ is left special there exists a unique $a \in \{1, 2, 3\}$ such that ua is left special, in other words a symmetric 3-hat language satisfies the hypothesis of Lemma 8. It follows that there exists two minimal infinite words $\mathcal{O}_1$ and $\mathcal{O}_2$ in $\{1, 2, 3\}^{\mathbb{N}}$ with $\mathcal{O}_i$ beginning in i and such that each prefix of $\mathcal{O}_i$ is left special. The $\mathcal{O}_i$ are called the *characteristic words*.

Knowing that a language is a symmetric 3-hat language, we are now able to build explicitely each $\mathcal{O}_i$: this algorithm is a variant of the *hat algorithm* given in [13] for constructing characteristic Sturmian or Arnoux-Rauzy words.

As in [13], to keep track of the bispecial prefixes of $\mathcal{O}_i$ we will accent the first letter of x with either an upper * or lower * according to the following rule: If u ends in 1, then x begins in either 2 or 3. The 2 is accented 2_* while the 3 is accented 3^*. If u ends in 2 then x begins in either 1 or 2. The 1 is accented 1_* and the 2 is accented 2^*. In this way, the bispecial prefixes of $\mathcal{O}_i$ are precisely those prefixes followed by a *-letter. When we copy the prefix x to the end of u we only accent the first letter of x and delete other accents which may be occurring in u. For example, suppose $\mathcal{O}_i$ begins in

$$\mathcal{O}_i = 1_*3^*13^*12_*22^*131313^*1221313131222^*13131\ldots$$

then the bispecial prefixes are (with the accents removed) $1_*, 1_*3^*1, 1_*3^*13^*1$, $1_*3^*13^*12_*2$, $1_*3^*13^*12_*22^*13131$, $1_*3^*13^*12_*22^*131313^*122131313122$. Suppose the next bispecial word u is $1_*3^*13^*12_*22^*131313^*1221313131222^*13131$ and $u2$ is left special. Then following Lemma 7, let v denote the longest prefix of $\bar{u} = u$ so that $2v$ is right special. Then v is right special (being a suffix of $2v$) and left special (being a prefix of u) and hence bispecial. It is readily verified that $v = 1_*3^*13^*1$ which is a palindrome. Thus following Lemma 8 the next bispecial word is obtained by copying the suffix $x = 2_*22^*131313^*1221313131222^*13131$ to the end of u. In doing so we delete all accents in x except the initial 2_* and obtain $u2 \models$
$1_*3^*13^*12_*22^*131313^*1221313131222^*131312_*2213131312213131312221313 1$.
Observe that x is the longest suffix of u beginning in 2_* and containing no other occurrences of 2_* or equivalently that v is the longest prefix of u followed by a 2_*. In general, a suffix x of u which begins in an accented letter $a \in \{1_*, 2^*, 2_*, 3^*\}$ and has no other occurrences of a is called a *cutting suffix* and denoted $u(a)$.

Proposition 10. *Let L be a symmetric 3-hat language. Then for every $k \geq 1$ there exist positive integers n_k and m_k and non-empty bispecial words u_k, $w_1^{(k)}, w_2^{(k)}, \ldots, w_{n_k}^{(k)}$, and $v_1^{(k)}, v_2^{(k)}, \ldots, v_{m_k}^{(k)}$ in the alphabet $\{1, 2, 3, 1_*, 2^*, 2_*, 3^*\}$ so that*

- *u_k is a prefix of $\mathcal{O}_1$, $|u_{k-1}| < |u_k|$ (where for $k = 0$ we take $u_0 = \varepsilon_\emptyset$), and $u_k \neq \bar{u_k}$.*
- *$w_1^{(1)} = 1_*$ and $v_1^{(1)} = 2_*^*$ where 2_*^* counts as a 2_* and 2^*.*
- *$w_1^{(k)}, w_2^{(k)}, \ldots, w_{n_k}^{(k)}$ and $v_1^{(k)}, v_2^{(k)}, \ldots, v_{m_k}^{(k)}$ are each palindromes and*

$$u_{k-1} \subset w_1^{(k)} \subset w_2^{(k)} \subset \cdots \subset w_{n_k}^{(k)} \subset u_k$$

and

$$\bar{u}_{k-1} \subset v_1^{(k)} \subset v_2^{(k)} \subset \cdots \subset v_{m_k}^{(k)} \subset \bar{u}_k.$$

$\{w_i^{(k)} \mid 1 \leq i \leq n_k\}$ are all the bispecial prefixes of $\mathcal{O}_1$ of length greater than $|u_{k-1}|$ and less than $|u_k|$ and $\{v_i^{(k)} \mid 1 \leq i \leq m_k\}$ are all the bispecial prefixes of $\mathcal{O}_2$ of length greater than $|\bar{u}_{k-1}|$ and less than $|\bar{u}_k|$.

- *If $u_{k-1}1$ is left special, then $w_{n_k}^{(k)}$ is the shortest word beginning in u_{k-1} such that $2w_{n_k}^{(k)}$ is right special and $v_{m_k}^{(k)}$ is the shortest word beginning in $\bar{u}_{k-1}$ such that $1v_{m_k}^{(k)}$ is right special.*
- *If $u_{k-1}2$ is left special, then $w_{n_k}^{(k)}$ is the shortest word beginning in u_{k-1} such that $3w_{n_k}^{(k)}$ is right special and $v_{m_k}^{(k)}$ is the shortest word beginning in $\bar{u}_{k-1}$ such that $2v_{m_k}^{(k)}$ is right special.*
- *If $u_{k-1}1$ is left special, then $w_1^{(k)} = u_{k-1}\bar{u}_{k-1}(1_*)$. For $2 \le i \le n_k$ $w_i^{(k)} = w_{i-1}^{(k)}w_{i-1}^{(k)}(3^*)$ if $w_{i-1}^{(k)}(3^*)$ exists, otherwise $w_i^{(k)} = w_{i-1}^{(k)}3^*w_{i-1}^{(k)}$. While $u_k = w_{n_k}^{(k)}v_{m_k}^{(k)}(2_*)$. Similarly, $v_1^{(k)} = \bar{u}_{k-1}u_{k-1}(2_*)$. For $2 \le i \le m_k$ $v_i^{(k)} = v_{i-1}^{(k)}v_{i-1}^{(k)}(2^*)$, and $\bar{u}_k = v_{m_k}^{(k)}w_{n_k}^{(k)}(1_*)$.*
- *If $u_{k-1}2$ is left special, then $w_1^{(k)} = u_{k-1}\bar{u}_{k-1}(2^*)$. For $2 \le i \le n_k$ $w_i^{(k)} = w_{i-1}^{(k)}w_{i-1}^{(k)}(2_*)$ and $u_k = w_{n_k}^{(k)}v_{m_k}^{(k)}(3^*)$. Similarly, $v_1^{(k)} = \bar{u}_{k-1}u_{k-1}(3^*)$ if it exists, otherwise $v_1^{(k)} = \bar{u}_{k-1}3^*u_{k-1}$. For $2 \le i \le m_k$ $v_i^{(k)} = v_{i-1}^{(k)}v_{i-1}^{(k)}(1_*)$, and $\bar{u}_k = v_{m_k}^{(k)}w_{n_k}^{(k)}(2^*)$.*

Let L be a symmetric 3-hat language. Let $(u_k)_{k\ge0}$, $(n_k)_{k\ge1}$ and $(m_k)_{k\ge1}$ be as in Proposition 10. For each $k \ge 1$ define $\epsilon_k \in \{+,-\}$ as follows: Let $a, b \in \{1_*, 2^*\}$ such that $u_{k-1}a$ and u_kb is left special. For $k = 1$, since u_0 is the empty word we set $a = 1$. Then set

$$\epsilon_k = \begin{cases} + \text{ if } a = b \\ - \text{ if } a \neq b \end{cases}$$

We call the sequence $(n_k, m_k, \epsilon_k)_{k\ge1}$ the *coding sequence* of L. In view of Proposition 10, this sequence contains the necessary information to construct all bispecial factors of L and hence L itself: in order to construct the language L we must specify for each bispecial word w the unique letter $c \in \{1, 2, 3\}$ such that wc is left special, and this information is contained in the coding sequence (n_k, m_k, ϵ_k). If w is bispecial and not a palindrome, then $w = u_k$ or $w = \bar{u}_k$ for some $k \ge 1$. In this case the information is coded in ϵ_k. On the other hand if w is a palindrome then $w = w_i^{(k)}$ for some $k \ge 1$ and $1 \le i \le n_k$ or $w = v_i^{(k)}$ for some $k \ge 1$ and $1 \le i \le m_k$. The n_k (respectively m_k) count the number of bispecial palindromes between u_{k-1} and u_k (respectively $\bar{u}_{k-1}$ and $\bar{u}_k$). If $w = w_i^{(k)}$ then n_k specifies the letter $c \in \{2_*, 3^*\}$ such that wc is left special since for one value of c (which by the last two points of Proposition 10 is determined from ϵ_{k-1}) the next bispecial prefix of $\mathcal{O}_1$ is a palindrome, while for the other value of c the next bispecial prefix of $\mathcal{O}_1$ is u_k and hence not a palindrome. Similarly if $w = v_i^{(k)}$ then c is coded in m_k.

Example 1. Let L be a symmetric 3-hat language whose coding sequence starts off as
$(3, 2, -), (1, 2, +), (2, 1, +)$. Then

$$\mathcal{O}_1 = 1_*3^*13^*12_*22^*131313^*1221313131222^*13131$$
$$2_*22131313122131313122213131 3^*122131313122$$

$$2^*131312221313131221313131222131 31\ldots$$

$$\begin{aligned}\mathcal{O}_2 = {} & 2_*^*2^*1_*31313^*1221_*313131222^*131313^*122131313122\\ & 2^*131312221313131221313131222131 31\\ & 3^*122131313122\ldots\end{aligned}$$

Example 2. Let L be a 3-SIET language with periodic coding sequence $(1,1,-),(1,1,-),(1,1,-)\ldots$. Then

$$\begin{aligned}\mathcal{O}_1 = {} & 1_*2_*2^*13^*121_*312212_*213122^*1221312131221\\ & 3^*121312212213121_*3122131213122122131221221312131221\ldots\end{aligned}$$

$$\begin{aligned}\mathcal{O}_2 = {} & 2_*^*1_*3^*122^*12_*213121_*312213^*12131221221312\\ & 2^*12213121312212_*213122122131213122131213122122131 2\ldots\end{aligned}$$

It can be shown that $\mathcal{O}_2$ is the fixed point of the morphism τ given by $1\mapsto 2$, $2\mapsto 2131$ and $3\mapsto 21$.

Another consequence of Proposition 10 is

Corollary 11. *If $(m_k,n_k,\epsilon_{k+1})_{k\geq 1}$ is the coding sequence of a symmetric 3-hat language, then $(n_k,\epsilon_{k+1})\neq(1,+)$ for infinitely many k and $(m_k,\epsilon_{k+1})\neq(1,+)$ for infinitely many k.*

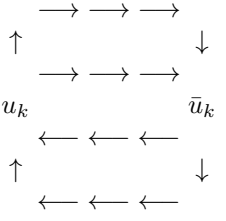
2.3 Return Words and Rauzy Graphs

Definition 12. For a word w in the language L, a *return word* of $w=w_1\ldots w_l$ is any word $w'=w'_1\ldots w'_{l'}$ such that there exist $i<j$ with $w=x_i\ldots x_{i+l-1}=x_j\ldots x_{j+l-1}$, $w\neq x_{i'}\ldots x_{i'+l-1}$ for every $i<i'<j$, and $w'=x_{i+l}\ldots x_{j+l-1}$.

We shall be interested in computing the return words of some remarkable words; for this, we need the following tools from the theory of *Rauzy graphs* (see [1] or [2]).

Definition 13. For two words w and w' of equal length l, w' is a *successor (in the Rauzy graph)* of w if there exists j such that $x_j\ldots x_{j+l-1}=w$, $x_{j+1}\ldots x_{j+l}=w'$. The Rauzy graph Γ_l is the graph whose vertices are all words of length l in the language of T, with an edge between each word w and each of its successors w', labelled with the last letter of w'.

Lemma 14. *k being fixed, let l be the length of u_k; the Rauzy graph Γ_l has the following shape:*



Proposition 15. *Let L be a symmetric 3-hat language. For every k, the k-th non-palindrome bispecial word u_k has exactly three return words, A_k, B_k and C_k, where $A_0 = 13$, $B_0 = 2$, $C_0 = 12$, and*

$$A_k = A_{k-1}^{n_k} C_{k-1} B_{k-1}^{m_k - 1},$$
$$B_k = B_{k-1} A_{k-1}^{n_k - 1} C_{k-1} B_{k-1}^{m_k - 1},$$
$$C_k = A_{k-1}^{n_k - 1} C_{k-1} B_{k-1}^{m_k - 1}$$

if $\epsilon_{k+1} = +1$,
and

$$A_k = B_{k-1} A_{k-1}^{n_k - 1} C_{k-1} B_{k-1}^{m_k - 1},$$
$$B_k = A_{k-1}^{n_k} C_{k-1} B_{k-1}^{m_k - 1},$$
$$C_k = B_{k-1} A_{k-1}^{n_k} C_{k-1} B_{k-1}^{m_k - 1}$$

if $\epsilon_{k+1} = -1$.

3 Languages of Symmetric Three-Interval Exchange Transformations

3.1 Preliminaries

Definition 16. A *symmetric three-interval exchange transformation* is the transformation defined by equation (1):

- $\mathcal{I}x = x + 1 - \alpha$ if $x \in [0, \alpha[$,
- $\mathcal{I}x = x + 1 - 2\alpha - \beta$ if $x \in [\alpha, \alpha + \beta[$,
- $\mathcal{I}x = x - \alpha - \beta$ if $x \in [\alpha + \beta, 1[$.

We say that $\mathcal{I}$, or equivalently (α, β), satisfies the *infinite distinct orbit condition* (or *i.d.o.c.* for short) of Keane [11] if for every integer p and q

- $p\alpha + q\beta \neq p - q$,
- $p\alpha + q\beta \neq p - q - 1$,
- $p\alpha + q\beta \neq p - q + 1$

This property is (strictly) weaker than the irrational independence of α, β and 1. The i.d.o.c. condition for $\mathcal{I}$ or (α, β) implies that $\mathcal{I}$ is *minimal* (every orbit is dense).

Definition 17. For every point x in $[0, 1[$, we define an infinite word $(x_n)_{n \in \mathbb{N}}$ by putting $x_n = i$ if $\mathcal{I}^n x$ falls into D_i, $i = 1, 2, 3$. This word is again denoted by x; we call it the *trajectory* of x.

If $\mathcal{I}$ satisfies the i.d.o.c. condition, the minimality implies that all trajectories have the same language, which we call the *language of* $\mathcal{I}$, and denote by $L(\mathcal{I})$.

Equivalently, $L(\mathcal{I})$ is the set of all words $w = w_0 w_1 \cdots w_n$ with $w_i \in \{1, 2, 3\}$ such that $\cap_{i=1}^{n} \mathcal{I}^{-i} D_{w_i}$ is a nonempty interval.

Lemma 18. *Let $\mathcal{I}$ be a symmetric three-interval exchange transformation satisfying the i.d.o.c. condition, and suppose that in the equation (1) $2\alpha < 1$ and $2\alpha + \beta > 1$. Then $L(\mathcal{I})$ is a symmetric 3-hat language.*

3.2 Description of the Language

Proposition 19. *Let $\mathcal{I}$ be a symmetric three-interval exchange transformation defined by equation (1), and satisfying the i.d.o.c. condition. We assume that* $2\alpha < 1$, $\alpha + \beta < 1$, $2\alpha + \beta > 1$.
Then $L(\mathcal{I})$ is the symmetric 3-hat language whose coding sequence $(n_k, m_k, \epsilon_{k+1})_{k\geq 1}$ *is defined in the following way:*
we start from $x_0 = \frac{1-\alpha-\beta}{1-\alpha}$ *and* $y_0 = \frac{1-2\alpha}{1-\alpha}$ *and define for* $k \geq 0$

$$x'_{k_+1} = \frac{y_k}{(x_k + y_k) - 1}$$

$$y'_{k+1} = \frac{x_k}{(x_k + y_k) - 1}$$

if $x_k + y_k > 1$,

$$x'_{k+1} = \frac{1 - y_k}{1 - (x_k + y_k)}$$

$$y'_{k+1} = \frac{1 - x_k}{1 - (x_k + y_k)}$$

if $x_k + y_k < 1$. *Then for* $k \geq 0$ m_{k+1} *is the integer part of* x'_{k+1}, n_{k+1} *is the integer part of* y'_{k+1}, , x_{k+1} *is the fractional part of* x'_{k+1}, , y_{k+1} *is the fractional part of* y'_{k+1}, *while* ϵ_{k+1} *is the sign of* $(1 - x_k - y_k)$.

4 Structure Theorems

4.1 Answer to Rauzy's Question: The Privileged Triangle

Definition 20. Let $D_0 \subset [0, 1[\times[0, 1[$ be the simplex bounded by the lines $y = 0$, $x = 0$, and $x + y = 1$. The *privileged triangle* D is the triangular region bounded by the lines $x = \frac{1}{2}$, $x + y = 1$, and $2x + y = 1$.

Lemma 21. *[9] The infinite sequence $(n_k, m_k, \epsilon_{k+1})_{k\geq 1}$ is the expansion, in the sense of Proposition 19, of some $(\alpha, \beta) \in D$ satisfying the i.d.o.c. if and only if n_k and m_k are positive integers, $\epsilon_{k+1} = \pm$, $(n_k, \epsilon_{k+1}) \neq (1, +)$ for infinitely many k and $(m_k, \epsilon_{k+1}) \neq (1, +)$ for infinitely many k.*

Theorem 22. *A language L is a symmetric 3-hat language if and only if $L = L(\mathcal{I})$, where $\mathcal{I}$ is a symmetric three-interval exchange transformation defined by equation (1) with $(\alpha, \beta) \in D$, and satisfying the i.d.o.c. condition.*

4.2 Answer to Rauzy's Question: Outside the Privileged Triangle

Proposition 23. *We define two mappings on* $[0,1[\times[0,1[$,

$$F(x,y) = \left(\frac{2x-1}{x}, \frac{y}{x}\right)$$

$$R(x,y) = (1-x-y, y)\,.$$

If $(\alpha,\beta) \in D_0$ *is not in* D *and is not on any of the rational lines* $p\alpha+q\beta = p-q$, $p\alpha+q\beta = p-q+1$, $p\alpha+q\beta = p-q-1$ *then there exists a unique finite sequence of integers* l_0, l_1, ..., l_k *such that* (α,β) *is in* $H^{-1}D$ *where* H *is a composition of the form* $R^t \circ F^{l_0} \circ R \circ F^{l_1} \circ R \ldots \circ R \circ F^{l_k} \circ R^s$, $s,t \in \{0,1\}$.

Proposition 24. *For* $(\alpha,\beta) \notin D$ *satisfying the i.d.o.c. condition, we let* H *be the function in Proposition 23 for which* $(\alpha,\beta) \in H^{-1}D$ *and put*

$$(\bar{\alpha},\bar{\beta}) = H(\alpha,\beta),$$

and define two morphisms, σ_F *by*

$$\begin{aligned} 1 &\to 1 \\ 2 &\to 21 \\ 3 &\to 31 \end{aligned}$$

and σ_R *by*

$$\begin{aligned} 1 &\to 3 \\ 2 &\to 2 \\ 3 &\to 1 \end{aligned}$$

and we define σ_H *by replacing, in the expression of* H, F *by* σ_F *and* R *by* σ_R.

Let $\mathcal{I}$ *be the three-interval exchange transformation defined by (1), and* $\mathcal{I}'$ *be the three-interval exchange transformation defined by putting bars on* α *and* β *in (1); then* $L(\mathcal{I})$ *is the image of* $L(\mathcal{I}')$ *by* σ_H.

4.3 S-Adic Presentation

Theorem 25. *Let* $\mathcal{I}$ *be a symmetric three-interval exchange transformation defined by equation (1) and satisfying the i.d.o.c. condition. For every* k, *the* k*-th non-palindrome bispecial word* u_k *of* $L(\mathcal{I})$ *has exactly three return words,* A_k, B_k *and* C_k, *given by the recursion formulas in Proposition 15.*

If $(\alpha\beta)$ *is in* D, $A_0 = 13$, $B_0 = 2$, $C_0 = 12$. *If* (α,β) *is not in* D, A_0 *is* $\sigma_H(13)$, B_0 *is* $\sigma_H(2)$, C_0 *is* $\sigma_H(12)$, *with* σ_H *as defined in Proposition 24. In all cases the lengths of* A_0 *and* B_0 *differ by* ± 1.

Corollary 26. *The symbolic system* (X,T) *coding* $\mathcal{I}$ *by the partition* $\{D_1,D_2,D_3\}$ *is the shift on all infinite words* x *such that for every* $s<t$ *there exists* k *such that* $x_s \ldots x_t$ *is a factor of* A_k, *given by the above formulas.*

This last result may be interpreted as the presentation of (X, T) as generated by the morphisms σ_k and σ'_k, where

$$\sigma_k a = a^{n_k} c b^{m_k - 1},$$
$$\sigma_k b = b a^{n_k - 1} c b^{m_k - 1},$$
$$\sigma_k c = a^{n_k - 1} c b^{m_k - 1},$$

and

$$\sigma'_k a = b a^{n_k - 1} c b^{m_k - 1},$$
$$\sigma'_k b = a^{n_k} c b^{m_k - 1},$$
$$\sigma'_k c = b a^{n_k} c b^{m_k - 1}.$$

A slight modification (by adding extra stages) would allow us to replace the countable family of the σ_k and σ'_k by a finite family of morphisms, and thus to define (X, T) as an *S-adic system*, see [8].

If, in the trajectory of a point x, we replace 1 by 0, 2 by 01, and 3 by 1, we get a Sturmian word, which is a concatenation of words A'_k, B'_k, C'_k given by the same recursion formulas but from $A_0 = B_0 = 01$, $C_0 = 001$; hence $A'_k = B'_k$ for all k, and this is a non-standard way to generate Sturmian words, close to the one used in [5].

References

1. P. ARNOUX, G. RAUZY: Représentation géométrique de suites de complexité $2n + 1$, *Bull. Soc. Math. France* **119** (1991), p. 199-215.
2. V. BERTHÉ, N. CHEKHOVA, S. FERENCZI: Covering numbers: arithmetics and dynamics for rotations and interval exchanges, *J. Analyse Math.* **79** (1999), p. 1-31.
3. J. CASSAIGNE: Complexité et facteurs spéciaux, *Bull. Belg. Math. Soc.* **4** (1997), p. 67-88.
4. E.M. COVEN, G.A. HEDLUND: Sequences with minimal block growth, *Math. Systems Theory* **7** (1972), p. 138-153.
5. A. del JUNCO: A family of counterexamples in ergodic theory, *Israël J. Math.* **44** (1983), p. 160-188.
6. G. DIDIER, Échanges de trois intervalles et suites sturmiennes, *J. Théor. Nombres Bordeaux,* **9** (1997), p. 463–478.
7. X. DROUBAY, J. JUSTIN, G. PIRILLO, Episturmian words and some constructions of de Luca and Rauzy, *Theoret. Comp. Sci.,* to appear.
8. S. FERENCZI: Rank and symbolic complexity, *Ergodic Th. Dyn. Syst.* 16 (1996), p. 663-682.
9. S. FERENCZI, C. HOLTON, L. ZAMBONI: The structure of three-interval exchange transformations I: an arithmetic study, to appear in *Ann. Inst. Fourier* **51** (2001).
10. A.B. KATOK, A.M. STEPIN: Approximations in ergodic theory, *Usp. Math. Nauk.* **22** (1967), p. 81-106 (in Russian), translated in *Russian Math. Surveys* **22** (1967), p. 76-102.
11. M.S. KEANE: Interval exchange transformations, *Math. Zeitsch.* **141** (1975), p. 25-31.

12. M. MORSE, G.A HEDLUND: Symbolic dynamics II. Sturmian trajectories, *Amer. J. Math.* **62** (1940), p. 1-42.
13. R. RISLEY, L.Q. ZAMBONI: A generalization of Sturmian sequences; combinatorial structure and transcendence, *Acta Arith.*, **95** (2000), 167–184.
14. G. RAUZY: Échanges d'intervalles et transformations induites, (in French), *Acta Arith.* **34** (1979), p. 315-328.
15. M.-L. SANTINI-BOUCHARD: Échanges de trois intervalles et suites minimales, (in French), *Theoret. Comput. Sci* **174** (1997), p. 171-191.
16. W.A. VEECH: The metric theory of interval exchange transformations I , II, III, *Amer. J. Math.* **106** (1984), p. 1331-1421.

Decision Questions Concerning Semilinearity, Morphisms, and Commutation of Languages*

Tero Harju[1], Oscar Ibarra[2]**, Juhani Karhumäki[3], and Arto Salomaa[4]

[1] Department of Mathematics and Turku Centre for Computer Science
University of Turku, FIN-20014 Turku, Finland
harju@utu.fi
[2] Department of Computer Science
University of California, Santa Barbara, CA 93106
ibarra@cs.ucsb.edu
[3] Department of Mathematics and Turku Centre for Computer Science
University of Turku, FIN-20014 Turku, Finland
karhumak@cs.utu.fi
[4] Turku Centre for Computer Science
Lemminkäisenkatu 14A, FIN-20520 Turku, Finland
asalomaa@cs.utu.fi

Abstract. Let $\mathcal{C}$ be a class of automata (in a precise sense to be defined) and $\mathcal{C}_c$ the class obtained by augmenting each automaton in $\mathcal{C}$ with finitely many reversal-bounded counters. We first show that if the languages defined by $\mathcal{C}$ are effectively semilinear, then so are the languages defined by $\mathcal{C}_c$, and, hence, their emptiness problem is decidable. This result is then used to show the decidability of various problems concerning morphisms and commutation of languages. We also prove a surprising undecidability result: given a fixed two element code K, it is undecidable whether a given context-free language L commutes with K, i.e., $LK = KL$.

Keywords: Reversal-bounded counters, context-free languages, combinatorics on words, commutation of languages, morphisms

1 Introduction

We shall consider various decision problems for families of languages accepted by classes of automata augmented with reversal-bounded counters. In particular, we generalize the decidability results of Parikh [P66] and Ibarra [I78] that have turned out to be very useful tools in proving other decidability results concerning language families. The classical result in [P66] states that the commutative image of a context-free language (*CF-language*) is semilinear, and that it can be effectively constructed from the pushdown automaton (*PDA*) defining the language.

* Supported under the grant 44087 of the Academy of Finland.
** Supported in part by NSF Grant IRI-9700370.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 579–590, 2001.

According to [I78] one can decide the emptiness problem for the languages accepted by pushdown automata augmented with reversal-bounded counters. The proof consists of showing that such automata accept only semilinear sets that are effectively constructible from the automaton.

In Section 3 we generalize the result in [I78]. We show that if the languages defined by a class $\mathcal{C}$ of automata are effectively semilinear, then so are the languages defined by the automata that are obtained from those of $\mathcal{C}$ by augmenting reversal-bounded counters. Several examples of $\mathcal{C}$ are given in Section 4. The established decidability result for semilinearity is exploited in Section 5 to show the decidability of the *multiple equivalence problem* of morphisms for a large family of languages. In this problem, one asks for a set of morphisms $h_1, h_2, \ldots, h_k$ and a language L whether each word $w \in L$ is identified by a pair of these morphisms, $h_i(w) = h_j(w)$ with $i \neq j$.

In Section 6 we apply the result to deterministic CF-languages L and regular codes R to show that it is decidable whether L can be expressed as $L = \cup_{i \in I} R^i$ for some set $I \subseteq N$. In contrast to this result, we show that the above question is undecidable for CF-languages and two element codes. From this later result it follows, by a result of [ChKO99], that if K is a given two element code, then it is undecidable whether a given CF-language L commutes with K, $LK = KL$.

We also present several open problems that involve the mentioned questions on semilinearity, language equivalence and commutation of languages.

2 Preliminaries

We refer to [Sa73] or [Har78] for the basic definitions on automata and languages, and to [ChK97] or [L83] to those on words.

For an alphabet Σ, denote by Σ^* the set of all words over Σ including the empty word, denoted by ε. Let $w \in \Sigma^*$ be a word. Then $|w|$ denotes the length of w, and $|w|_a$ denotes the number of occurrences of the letter $a \in \Sigma$ in w.

For a language $L \subseteq \Sigma^*$, let $\overline{L}$ denote its complement, $\Sigma^* \setminus L$, and let $L^+ = \{w_1 w_2 \ldots w_i \mid w_i \in L, i \geq 1\}$ be its Kleene closure. Denote $L^* = L^+ \cup \{\varepsilon\}$. A language L is a *code*, if each word $w \in L^+$ has a unique factorization $w = w_1 w_2 \ldots w_n$ in terms of elements $w_i \in L$.

The *shuffle* $u \sqcup v$ of two words $u, v \in \Sigma^*$ consists of the words $u_1 v_1 \ldots u_k v_k$, where $u = u_1 u_2 \ldots u_k$ and $v = v_1 v_2 \ldots v_k$ for some $u_i, v_i \in \Sigma^*$. For languages L and K, their *shuffle* is the language $L \sqcup K = \bigcup_{u \in L, v \in K} u \sqcup v$.

In this paper, a *language family* will mean a class $\mathfrak{L}$ of languages that are accepted (or defined) by the automata from a class of automata $\mathcal{M}$. That is, if $\mathfrak{L}$ is a language family, then each of its element $L \in \mathfrak{L}$ is specified by $L = L(M)$ for some $M \in \mathcal{M}$. Let P be a (binary) language operation, i.e., P maps two languages L_1 and L_2 to a language $P(L_1, L_2)$. The family $\mathfrak{L}$ is *effectively closed* under the operation P, if for any $L_i = L(M_i)$ with $M_i \in \mathcal{M}$ for $i = 1, 2$, $P(L_1, L_2) \in \mathfrak{L}$ and one can effectively construct an $M \in \mathcal{M}$ such that $L(M) = P(L_1, L_2)$.

We study language families accepted by multitape Turing machines, where the worktapes have restricted behaviour. The storage types will be variants of the

pushdown storages and queues, the behaviour of which may be further restricted. If $\mathcal{M}$ is a class of automata, then $\mathcal{L}(\mathcal{M}) = \{L(M) \mid M \in \mathcal{M}\}$ denotes the family of languages accepted by the automata in $\mathcal{M}$. E.g., $\mathcal{L}(PDA)$ denotes the family of CF-languages, the languages accepted by 1-way pushdown automata (PDAs).

Consider a class $\mathcal{M}$ of automata, where each $M \in \mathcal{M}$ is a nondeterministic finite automaton possibly augmented with a data structure (consisting of Turing tapes with restricted behaviour). Formally, $M = \langle Q, \Sigma, q_0, F, D, \delta \rangle$, where Q is the finite state set, Σ is the input alphabet, q_0 is the start state, F is the set of accepting states, D is the data structure (consisting of k worktapes), and δ is the (multiple valued) transition function. The automaton M has finitely many transitions (i.e., moves) of the form $(p, act) \in \delta(q, a, loc)$, where $q \in Q$ is the current state, $a \in \Sigma \cup \{\varepsilon\}$ is the input read, and $loc = (a_1, \ldots, a_k)$ is the "local" portion of the data structure D that affects the move, where $a_i = \varepsilon$ or a_i is the letter that is read on the ith worktape; $p \in Q$ is the state entered by M, and $act = (\alpha_1, \alpha_2, \ldots, \alpha_k)$ is the action that M performs on the worktapes.

For example, if D consists of a pushdown stack and a queue, then $loc = (a, b)$, where a is the top symbol of the stack and b the symbol in the front of the queue or ε if the queue is empty; and $act = (\alpha_1, \alpha_2)$, where α_1 pops the top symbol and pushes a word (possibly empty) onto the stack, and α_2 deletes the front of the queue (if $b \neq \varepsilon$) and possibly adds a symbol to the rear of the queue.

3 Semilinearity and Augmented Counters

Let $\mathbb{N} = \{0, 1, \ldots\}$, and k be a positive integer. A subset $S \subseteq \mathbb{N}^k$ is *linear*, if there exist vectors $v_0, \ldots, v_t \in \mathbb{N}^k$ such that $S = \{v \mid v = v_0 + a_1 v_1 + \cdots + a_t v_t,\ a_i \in \mathbb{N}\}$. The vectors v_0 (referred to as the *constant vector*) and $v_1, v_2, \ldots, v_t$ (referred to as the *periods*) are called the *generators* of the linear set S. The set $S \subseteq \mathbb{N}^k$ is *semilinear* if it is a finite union of linear sets. Every finite subset of $\mathbb{N}^k$ is semilinear. It is also clear that the semilinear sets are closed under (finite) union.

Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$. Define the *Parikh map* of $w \in \Sigma^*$ to be $\psi(w) = (|w|_{a_1}, \ldots, |w|_{a_n})$. For a language $L \subseteq \Sigma^*$, let $\psi(L) = \{\psi(w) \mid w \in L\}$. The language L is *semilinear* if $\psi(L)$ is a semilinear set.

Obviously, if the languages in a family $\mathcal{L}$ $(= \mathcal{L}(\mathcal{M}))$ are effectively semilinear (that is, for each $M \in \mathcal{M}$, the semilinear set $S = \psi(L)$ can be effectively constructed), then the emptiness problem for $\mathcal{L}$ is decidable.

The following result was shown in [P66].

Theorem 1. *Let M be a 1-way nondeterministic pushdown automaton. Then $\psi(L(M))$ is a semilinear set effectively computable from M.*

We can augment a PDA with finitely many *reversal-bounded counters*, i.e., each counter can be tested for zero and can be incremented/decremented by one, but the number of alternations between nondecreasing mode and nonincreasing mode in any computation is bounded by a given constant. For example, a counter whose values change according to the pattern 0 1 1 2 3 4 $\underline{4\ 3}$ 2 1 $\underline{0\ 1}$ $\underline{1\ 0}$ is 3-reversal (here the reversals are underlined).

The next result, which generalizes Theorem 1, was proved in [I78].

Theorem 2. *Let M be a 1-way nondeterministic pushdown automaton augmented with finitely many reversal-bounded counters. Then $\psi(L(M))$ is a semilinear set effectively computable from M.*

For a class $\mathcal{C}$ of automata, denote by $\mathcal{C}_c$ the class obtained by augmenting each automaton in $\mathcal{C}$ with finitely many reversal-bounded counters. The proof technique in [I78] easily generalizes for the following result.

Theorem 3. *Let $\mathcal{C}$ be a class of automata. If the languages in $\mathcal{L}(\mathcal{C})$ are effectively semilinear, then so are the languages in $\mathcal{L}(\mathcal{C}_c)$. Hence, the emptiness problem for the languages defined by $\mathcal{C}_c$ is decidable.*

Proof. Let $M_c \in \mathcal{C}_c$ with k reversal-bounded counters. We may assume, without loss of generality, that for the accepted inputs, each counter starts and ends with zero value. Thus, each counter makes an odd number of reversals. In fact, we can assume that each counter makes exactly one reversal, since a counter that makes r reversals can be converted to $(r+1)/2$ counters, each making one reversal (one counter for each change from a nonincremental to an incremental move). We show that $\psi(L(M_c))$ is effectively semilinear. We give the proof for $k = 1$, i.e., M_c has only one 1-reversal augmented counter. The proof for the general case follows inductively from this.

Let $\Sigma = \{a_1, \ldots, a_n\}$ be the input alphabet of M_c, and $L_c = L(M_c)$ be the language accepted by M_c. Denote $\Sigma_1 = \Sigma \cup \{d, e\}$, where $d, e \notin \Sigma$, and define a morphism $\varphi\colon \Sigma_1^* \to \Sigma^*$ by $\varphi(a) = a$ for $a \in \Sigma$ and $\varphi(d) = \varepsilon = \varphi(e)$.

We construct an automaton M in $\mathcal{C}$ whose input alphabet is Σ_1 accepting a language $L \subseteq \Sigma^* \sqcup d^*e^*$. The automaton M works as follows on an input word $w \in \Sigma_1^*$. M simulates the computation of M_c on $w' = \varphi(w)$ and uses the letters d and e in w to simulate the actions of the counter: an increment "+1" (resp. decrement "-1") corresponds to reading a symbol d (resp. e) on the input. Simultaneously M checks that $w \in \Sigma^* \sqcup d^*e^*$. At some point during the simulation, M guesses that the number of e's read is equal to the number of d's (corresponding to the counter becoming zero); M continues the simulation, making sure that there are no more d's and e's encountered, and accepts if M_c accepts.

Therefore, $L \subseteq \Sigma^* \sqcup d^*e^*$, and for each $w' \in L_c$, there exists a word $w \in L$ such that $w \in \Sigma^* \sqcup \{d^i e^i \mid i \geq 0\}$ and $\varphi(w) = w'$.

By assumption, the Parikh map $\psi(L)$ of L is an effectively computable semilinear set S_1. Let S_2 be the semilinear set $\{(i_1, \ldots, i_n, k, k) \mid i_j, k \in \mathbb{N}\}$, where the last two coordinates correspond to symbols d and e, respectively. The set $S_3 = S_1 \cap S_2$ is semilinear (here intersecting with S_2 essentially gets rid of the "non-valid computations"). The set S_3 is effectively computable, since semilinear sets are effectively closed under intersection [G66]. Now the semilinear set S corresponding to the Parikh map $\psi(L_c)$ of L_c is obtained from S_3 by simply removing the last two coordinates of the tuples.

We continue the proof of the previous theorem to show

Theorem 4. *Let $\mathcal{C}$ and $\mathcal{D}$ be two classes of automata. If $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{D})$, then also $\mathcal{L}(\mathcal{C}_c) = \mathcal{L}(\mathcal{D}_c)$.*

Proof. Let $L_c = L(M_c) \subseteq \Sigma^*$ for $M_c \in \mathcal{C}_c$. We adopt the notations, conventions and the construction of the proof of Theorem 3. In particular, M_c has only one 1-reversal augmented counter. Let $K = \Sigma^* \sqcup \{d^n e^n \mid n \geq 0\}$, and let $L = L(M)$, where $M \in \mathcal{C}$ is the simulating automaton. Clearly, $L_c = \varphi(L \cap K)$, and, by assumption, there is an $M' \in \mathcal{D}$ such that $L = L(M')$. Hence $L_c = \varphi(L(M') \cap K)$. Now $L(M'_c) = L \cap K$ for some $M'_c \in \mathcal{D}_c$, since for the intersection with K we can use one new 1-reversal counter. Finally, $L(M''_c) = \varphi(L \cap K)$ for some $M''_c \in \mathcal{D}_c$, since for the morphic image w.r.t. φ, no new counters are needed.

Note that Theorem 4 does not hold in converse. Indeed, for this it suffices to choose $\mathcal{C}$ as the class of all finite automata, and $\mathcal{D} = \mathcal{C}_c$.

A *simple shuffle language* is a language of the form $\Sigma^* \sqcup \{d^n e^n \mid n \geq 0\}$, for some alphabet Σ and distinct symbols d, e.

Corollary 1. *$\mathcal{L}(\mathcal{C}_c)$ is the smallest class of languages containing $\mathcal{L}(\mathcal{C})$ that is closed under morphisms and intersections with simple shuffle languages.*

4 Examples of Automata Classes

In this section we give examples of classes of automata with reversal-bounded counters that accept only semilinear languages. Hence the emptiness problem is decidable for the defined languages.

A *PCA* is a 1-way nondeterministic pushdown automaton augmented with reversal-bounded counters. By Theorem 1, every CF-language $L \in \mathcal{L}(PDA)$ is semilinear. Hence, by Theorem 3, the languages in $\mathcal{L}(PCA)$ are semilinear and, therefore their emptiness problem is decidable. (This is Theorem 2.)

A *CA* is a 1-way nondeterministic finite automaton augmented with reversal-bounded counters. Thus, it is a PCA without a pushdown stack. A *2CA* is a two-way nondeterministic finite automaton with end markers augmented with reversal-bounded counters.

We note that the emptiness problem for 2CAs is undecidable, even when the automaton is deterministic and it has only two reversal-bounded counters [I78].

A *finite-crossing 2CA* is a 2CA where the number of times the input head crosses the boundary between any two adjacent cells of the input tape (including the end markers) is bounded by a given constant. Every finite-crossing 2CA can effectively be converted to a (1-way) CA [GuI81]. (However, the nondeterminism is essential here, see [IJTW95].) Therefore,

Corollary 2. *The languages accepted by finite-crossing 2CAs are semilinear, and hence their emptiness problem is decidable.*

In contrast to the undecidability for 2CAs with two reversal-bounded counters, the emptiness problem is decidable for deterministic 2CAs with only one reversal-bounded counter [IJTW95]. These automata can accept fairly complex languages, e.g., the language $L = \{0^k 1^n \mid k \text{ divides } n\}$ that is not semilinear.

Problem 1. *Is the emptiness problem decidable for nondeterministic 2CAs with only one reversal-bounded counter?*

We extend a PCA as follows. An *MPA* has multiple pushdown stacks, ordered by name, $S_1, \ldots, S_m$, such that it can only read the top symbol of the *first* nonempty stack. (Without this restriction, an MPA can simulate a Turing machine.) An *MPCA* is an MPA augmented with reversal-bounded counters. A move of an MPCA depends only on the current state, the input symbol (or ε), the status of the counters (zero or nonzero), and the top symbol of the first nonempty stack, say S_i; initially, the first stack contains a starting top symbol Z_0 and all other stacks are empty. The action taken in a move consists of the input being consumed, each counter being updated, the top symbol of S_i being popped and a word (possibly ε) being pushed onto each stack, and the next state being entered. An MPA can be quite powerful. E.g., the language $\{x^k \mid x \in \{a,b\}^*\}$, where k is a fixed integer, can be accepted by an MPA. However, it was shown in [BCCC96] that MPAs accept semilinear languages only, and thus, by Theorem 3, we have the following result, also observed recently in [D00].

Corollary 3. *The languages in $\mathcal{L}(MPCA)$ are semilinear, and their emptiness problem is decidable.*

A *TA* is a 1-way nondeterministic finite automaton with a finite-crossing two-way read/write worktape, i.e., the number of times the head crosses the boundary between any two adjacent cells of the worktape is bounded by a constant, independent of the computation. A *TCA* is a TA with augmented reversal-bounded counters [IBS00]. Note that in a TA there is no bound on how long the head of the worktape can remain on a cell.

Corollary 4 will state that TCAs accept only semilinear languages. Note that if the worktape is not finite-crossing, then the automaton is equivalent to a Turing Machine. A TCA can be quite powerful. Let $L_0 \subseteq \Sigma^*$ over $\Sigma = \{a,b,c,d\}$ consist of the words $x = x_1 a c^{i_1} b x_2 \ldots x_n a c^{i_n} b x_{n+1}$, where $n \geq 0$ and $x_i \notin \Sigma^* a \Sigma^* b \Sigma^*$ such that $|x|_d = \sum_{j=1}^{n} c_j$. It can be shown that there exists a deterministic TCA M that accepts $L = \{x\#x \mid x \in L_0\}$, where $\# \notin \Sigma$. Here M has one counter, M is 9-crossing, although the worktape is not finite-turn.

We shall prove the semilinearity of the language in $\mathcal{L}(TA)$ using the following two lemmas, where we say that a TA M is *non-sitting*, if in any computation of M the read/write head always moves left or right of a cell in every step.

Lemma 1. *Let M_1 be a TA (i.e., without counters). We can effectively construct a TA M_2 such that $L(M_2) = L(M_1)$ and M_2 is non-sitting.*

Proof. Let # be a new 'dummy' symbol. Now M_2 begins the simulation of M_1 by writing nondeterministically a finite-length sequence of #'s on the worktape; M_2 simulates M_1, but whenever M_1 writes a symbol on a *new* tape cell, then M_2 writes nondeterministically to the right of this cell a finite-length sequence of #'s. Thus, at any time, the worktape contains a word where every pair of non-dummy symbols is separated by a word of #'s. During the simulation, M_2 uses/moves on the #'s to simulate the sitting moves of M_1, which is possible if there are enough #'s between any pair of non-dummy symbols. To simulate a nonsitting move of M_1, M_2 may need to "skip over" the #'s to get to the correct non-dummy symbol. Clearly, M_2 is non-sitting and accepts $L(M_1)$.

Lemma 2. *Let M be a TA. Then $\psi(L(M))$ is an effective semilinear set.*

Proof. By Lemma 1, we assume that M is non-sitting. We may also assume that M's worktape is 1-way infinite and that a blank cell when visited is re-written by a non-blank symbol, unless the automaton halts directly upon visiting the cell. We number the worktape cells by $1, 2, \ldots$ from left to right.

Consider an accepting computation of M on an input x such that M uses n worktape cells. By assumption, M accepts with its read/write head on cell n, which is blank. Now look at the cell p of the worktape, $p = 1, 2, \ldots, n$. In the computation, cell $p < n$ may be visited several times, but cell n is visited exactly once, on acceptance. Let $t_1, \ldots, t_m$ be the times M visits p.

Corresponding to the time sequence $(t_1, \ldots, t_m)$ associated with p, we define a *crossing vector* $R = (I_1, \ldots, I_m)$, where for each i, $I_i = (d_1, q_1, r_1, r_2, d_2)$, $d_1 \in \{-1, +1\}$ is the direction from which the head entered p at time t_i; q_1 is the state when M entered p; r_1 is the instruction that was used in the move above; r_2 is the instruction that was used at time $t_i + 1$ when M left p; $d_2 \in \{-1, +1\}$ is the direction to which M left p at time $t_i + 1$. Note that instruction r_2 specifies the input $a_i \in \Sigma \cup \{\varepsilon\}$ that M reads at time $t_i + 1$. Denote $\gamma(R) = a_1 \ldots a_m$.

We construct a 1-way nondeterministic finite automaton M' that simulates an accepting computation of M on w by nondeterministically guessing the sequence of crossing vectors $R_1, \ldots, R_n$ as M' processes the worktape from left to right, making sure that R_j and R_{j+1} are compatible for $1 \leq j \leq n - 1$. During the simulation, M' also checks that its input is $v = \gamma(R_1)\gamma(R_2)\ldots\gamma(R_{n-1})$. Clearly, v is a permutation of w, and $\psi(v) = \psi(w)$. Hence, by Theorem 1, $\psi(L(M)) = \psi(L(M'))$ is a semilinear set.

By Theorem 3, we have the following corollary.

Corollary 4. *The languages in $\mathcal{L}(TCA)$ are semilinear, and their emptiness problem is decidable.*

5 The Equivalence Problems for Sets of Maps

In this section $\mathcal{C}$ denotes a class of automata that satisfies the requirements of Theorem 3. In particular, the languages accepted by the automata $M \in \mathcal{C}$ are effectively semilinear.

Let $g, h\colon \Sigma^* \to \Delta^*$ be two mappings for the alphabets Σ and Δ. The *equality set* of g and h is defined to be $E(g, h) = \{w \in \Sigma^* \mid g(w) = h(w)\}$.

The problem whether $E(g, h) = \{\varepsilon\}$ is undecidable for the equality sets of morphisms. This problem is known as the *Post Correspondence Problem*.

Let $\mathcal{F}$ be a family of mappings between word monoids, and let $\mathcal{L}$ be a language family. The *equivalence problem of maps from $\mathcal{F}$ on $\mathcal{L}$* is the problem whether $L \subseteq E(g, h)$ for $L \in \mathcal{L}$ and $g, h \in \mathcal{F}$.

The equivalence problem of morphisms is decidable on CF-languages [CS78]. However, by the undecidability of the universe problem, the problem whether $L = E(h, g)$ is undecidable for morphisms and CF-languages.

In the *multiple equivalence problem of* $\mathcal{F}$ *on* $\mathcal{L}$ we ask for a given language $L \in \mathcal{L}$, and a finite set (g_i, h_i) of pairs of maps $g_i, h_i \in \mathcal{F}$ for $i = 1, 2, \ldots, k$, whether $L \subseteq \bigcup_{i=1}^{k} E(g_i, h_i)$.

Theorem 5. *The multiple equivalence problem of morphisms is decidable on languages accepted by automata in* $\mathcal{C}$.

Proof. Let $L = L(M)$, (g_i, h_i), $i = 1, 2, \ldots, k$, where $M \in \mathcal{C}$, $L \subseteq \Sigma^*$, and $g_i, h_i \colon \Sigma^* \to \Delta_i^*$ for each i. Denote $L' = \{w \in L \mid g_i(w) \neq h_i(w) \text{ for all } i\}$. Clearly, each $w \in L$ satisfies $g_i(w) = h_i(w)$ for some i if and only if $L' = \emptyset$. We show that $L' = L(M')$ for an automaton $M' \in \mathcal{C}_c$, obtained from a machine in $\mathcal{C}$ by augmenting it with reversal-bounded counters. Hence, by Theorem 3, the emptiness problem of the associated languages L' is decidable.

The automaton M' uses 2k counters $c_{11}, c_{12}, \ldots, c_{k1}, c_{k2}$. On an input w, the counters c_{i1} and c_{i2} are used to find a discrepancy in $g_i(w)$ and $h_i(w)$, that is, a position p_i, where the words $g_i(w)$ and $h_i(w)$ differ from each other. Now M' simulates M and at the same time applies, simultaneously for all $i = 1, 2, \ldots, k$, the morphisms g_i and h_i on the input w by using the counter c_{i1} (resp. c_{i2}) to guess and store the position p_{i1} in $g_i(w)$ (resp. p_{i2} in $h_i(w)$) where a discrepancy might occur. Therefore, either $g_i(w) = u_1 b_{i1} v_1$, where $|u_1| = p_{i1} - 1$, or $b_{i1} = \varepsilon$ and $p_{i1} > |g_i(w)|$; and similarly for h_i, either $h_i(w) = u_2 b_{i2} v_2$, where $|u_2| = p_{i2} - 1$, or $b_{i2} = \varepsilon$ and $p_{i2} > |h_i(w)|$. M' records the symbols $b_{i1}, b_{i2} \in \Delta_i \cup \{\varepsilon\}$, and continues the simulation of M until the input word w is consumed. So far, M' has just increased the counters, and has made no reversals. Now, M' checks that for all i, $b_{i1} \neq b_{i2}$, and if M accepts the word w, then M' verifies that for all i, $p_{i1} = p_{i2}$ by simultaneously decrementing the counters c_{i1} and c_{i2} and checking that they reach zero at the same time. Finally, M' accepts when all the counter checks succeed. Clearly, M' accepts L', and the claim follows.

As a corollary to Theorem 5, we obtain

Corollary 5. *It is decidable for finitely many morphisms* $h_1, \ldots, h_k$ *and a language* L *accepted by an automaton in* $\mathcal{C}$ *whether for each word* $w \in L$, $h_i(w) = h_j(w)$ *for some* $i \neq j$.

The proof of Theorem 5 generalizes to the case of mappings computed by deterministic generalized sequential machines (*gsms*) with reversal-bounded counters (and with or without accepting states). The following result improves a result of [CS78].

Theorem 6. *The multiple equivalence problem of the mappings computed by the deterministic gsms with reversal-bounded counters is decidable on the languages accepted by automata in* $\mathcal{C}$.

There are natural variations of the above results, e.g., one might be interested in deciding, given $L, h_1, \ldots, h_k$, and $m \leq k$, whether for each $w \in L$, there are at least m mappings h_i that map w to the same word.

Problem 2. *For which language families is the following problem decidable? Let* L *be a language,* h *be a nondeterministic gsm mapping and* g *be a deterministic gsm mapping. Does there exist a word* $w \in L$ *such that* $g(w) \notin h(w)$*?*

Note that the above problem is undecidable when both h and g are nondeterministic gsm mappings, in fact surprisingly, even when both h and g are finite substitutions and L is fixed to be the bounded language ab^*c [KL00].

6 Problems on Commutation

In the *commutation problem* for a language family $\mathfrak{L}$ we ask whether $LK = KL$ for two given languages $L, K \in \mathfrak{L}$. In the *equivalence problem* for a family $\mathfrak{L}$ we ask for given languages $L, K \in \mathfrak{L}$ whether $L = K$.

Theorem 7. *Let $\mathfrak{L}$ be a language family containing the singleton sets and effectively closed under concatenation. Then the commutation problem is decidable for $\mathfrak{L}$ if and only if the equivalence problem is decidable for $\mathfrak{L}$.*

Proof. If $L, K \in \mathfrak{L}$, then also $LK, KL \in \mathfrak{L}$, and therefore if the equivalence problem is decidable for $\mathfrak{L}$, so is the commutation problem. Assume then that the commutation problem is decidable for $\mathfrak{L}$. Let $L, K \in \mathfrak{L}$, and let $\#$ be a symbol not in the alphabet of L and K. Then the equivalence $L = K$ can be decided, since $(L\#)(K\#) = (K\#)(L\#)$ if and only if $L = K$.

It follows that the commutation problem is undecidable for CF-languages. In fact, it is undecidable for languages accepted by finite automata augmented with a 1-reversal counter, since for these automata, even with only 4 states the universe problem, and hence the equivalence problem, is undecidable [HalH98].

The equivalence problem is decidable in $\mathfrak{L}(DPDA)$ [Se97] (see also [St00]), that is, for the languages accepted by deterministic PDAs. Now, $\mathfrak{L}(DPDA)$ is not closed under concatenation (even from the left by two element sets, see [Har78]). However, $\mathfrak{L}(DPDA)$ is effectively closed under *marked concatenation*: if $L, K \in \mathfrak{L}(DPDA)$, then also $L\#K \in \mathfrak{L}(DPDA)$, where $\#$ is a new symbol. Therefore, by the proof of Theorem 7, if the commutation problem is decidable for deterministic CF-languages, a direct proof of this is likely to be very difficult.

Problem 3. *Is the commutation problem decidable in $\mathfrak{L}(DPDA)$?*

Problem 4. *For which language families is the commutation problem, $LK = KL$, decidable when K is a finite language?*

A *deterministic finite-turn 2CA* is a special case of the deterministic finite-crossing 2CA, where the head makes at most a fixed number of turns on the input tape. The equivalence problem for these automata, in fact, for deterministic finite-crossing 2CAs, is decidable [I78,GuI81,IJTW95]. Moreover, it can be shown that, given a finite set K and a language L accepted by a deterministic finite-turn 2CA, we can effectively construct deterministic finite-turn 2CAs accepting KL and LK. Hence Problem 4 is decidable for deterministic finite-turn 2CAs.

We shall now show that Problem 4 has a negative answer for CF-languages. Indeed, we prove that the commutation problem is undecidable for $K = \{a, b\}$ and CF-languages L. For the proof we need the following result from [ChKO99].

Lemma 3. *Let K be a two element code and L be any language. Then $LK = KL$ if and only if there exists a subset $I \subseteq \mathbb{N}$ such that*

$$L = \bigcup_{i \in I} K^i. \tag{1}$$

We note that if (1) holds for a code K and a language L, then the set I is uniquely determined, since $K^i \cap K^j = \emptyset$ for all $i \neq j$.

Theorem 8. *Let K be a fixed two element code. It is undecidable whether for a context-free language L there exists a set I such that $L = \cup_{i \in I} K^i$.*

Proof. Let $K = \{x_1, x_2\} \subseteq \Sigma^*$. Let $\Delta = \{a_1, a_2\}$ be an alphabet, and define a bijective morphism $\varphi\colon \Delta^* \to \Sigma^*$ by $\varphi(a_i) = x_i$ for $i = 1, 2$.

Firstly, it is undecidable for CF-languages $L \subseteq \Sigma^*$ whether or not $L = K^*$. Indeed, for any CF-language L', $L' = \Delta^*$ if and only if $\varphi(L') = K^*$, and the claim follows, since $L = \varphi(L')$ is effectively context-free, and the universe problem is undecidable for CF-languages.

Suppose contrary to the claim of the theorem that the existence of a set I of powers can be decided. We derive a contradiction from the undecidability of the equivalence problem $L = K^*$. Let $L \subseteq \Sigma^*$ be any CF-language. If there does not exist a set I such that $L = \cup_{i \in I} K^i$, then trivially $L \neq K^*$. Suppose then that such an I exists. We show that I is a semilinear subset of $\mathbb{N}$ (that is, I is ultimately periodic) and it can be effectively constructed from L. We have $\varphi(\varphi^{-1}(L)) = L$, since $L \subseteq K^*$, and hence $\varphi^{-1}(L) = \bigcup_{i \in I} \Delta^i$. Because $\varphi^{-1}(L)$ is effectively context-free, the length set I of $\varphi^{-1}(L)$ is semilinear, and it can be effectively constructed from $\varphi^{-1}(L)$, and thus from L. It follows that L is effectively regular, and therefore we can decide whether or not $L = K^*$. This contradicts the undecidability claim in the beginning of the proof.

By Lemma 3, we have the following corollary.

Corollary 6. *Let K be a fixed two element code. It is undecidable for context-free languages L whether or not $KL = LK$.*

Theorem 8 suggests the following general problem.

Problem 5. *For which language families $\mathcal{L}$ is the following problem decidable: Given a (possibly infinite) regular code R and a language $L \in \mathcal{L}$, does there exist a set I such that $L = \cup_{i \in I} R^i$?*

By Theorem 8, Problem 5 is undecidable for CF-languages. We show now that the problem is decidable for deterministic CF-languages.

If a set R is a code, then $R^i \cap R^j = \emptyset$ for all $i \neq j$, and therefore we have

Lemma 4. *Let $R \subseteq \Sigma^*$ be regular code and $L \subseteq \Sigma^*$ a language. Then there exists a set I such that $L = \cup_{i \in I} R^i$ if and only if $L \subseteq R^*$ and, for all $i \geq 0$,*

$$L \cap R^i \neq \emptyset \implies \overline{L} \cap R^i = \emptyset. \tag{2}$$

Theorem 9. *It is decidable for deterministic context-free languages L and regular codes R whether or not $L = \cup_{i \in I} R^i$ for some set I.*

Proof. The containment problem $L \subseteq R^*$ is decidable for CF-languages, since R^* is a regular language. Assume thus that $L \subseteq R^*$ holds. For (2), we recall that the deterministic CF-languages are effectively closed under complementation. Let $L = L(M_1)$, $\overline{L} = L(M_2)$, and $L(A) = R^*$, for deterministic PDAs M_1, M_2, and a finite automaton A. Let # be a new symbol. We construct a nondeterministic PDA M augmented with one 1-reversal counter that accepts a word $u\#v \in \Sigma^*\#\Sigma^*$ if and only if $u \in L \cap R^i$ and $v \in \overline{L} \cap R^i$ for some i. Therefore $L(M) = \emptyset$ if and only if L satisfies the condition (2). Since the emptiness problem is decidable for the languages accepted by PCAs, the claim follows.

Let then $w = u\#v$ be an input word. Then M simulates M_1 and A in parallel on u, and M checks that $u \in L$ and $u \in R^i$ for some i, recording i in the counter. Note that i is unique, because R is a code. In this part M needs to be nondeterministic. Then M simulates M_2 and A on v and checks that $v \in \overline{L}$ and $v \in R^i$ for the same i that was recorded in the counter. This M does by decrementing the counter. Finally, M accepts if M_2 accepts.

Theorem 9 generalizes in many different ways to larger language families. We note that the condition (2) is decidable also in the following cases: (1) L is accepted by a deterministic MPCA and R is accepted by a CA. (2) L is accepted by a deterministic CA and R is accepted by an MPCA. (3) L is accepted by deterministic finite-crossing 2CA (since this class is effectively closed under complementation [Gul81]) and R is accepted by a CA. (4) L and R are accepted by deterministic 2CAs with only one reversal-bounded counter, as this class is effectively closed under complementation [IJTW95].

The condition (2) is undecidable when R is a deterministic context-free code as is shown by the general result in Theorem 10. The *disjointness problem* for a family $\mathfrak{L}$ is the problem whether $L_1 \cap L_2 = \emptyset$ for two languages $L_1, L_2 \in \mathfrak{L}$.

Theorem 10. *Let $\mathfrak{L}$ be a language family that is effectively closed under concatenation and union with singleton sets. If the disjointness problem is undecidable for $\mathfrak{L}$, then so is the condition (2) for codes $L, R \in \mathfrak{L}$.*

Proof. Given $L_1, L_2 \in \mathfrak{L}$ over the alphabet Σ, define L and R over the alphabet $\Sigma \cup \{\#, \$\}$ by: $L = L_1\#$ and $R = L_2\# \cup \{\$\}$. Clearly, both L and R are codes in $\mathfrak{L}$. Now, there exists an i such that $L \cap R^i \neq \emptyset$ and $\overline{L} \cap R^i \neq \emptyset$ if and only if $L_1 \cap L_2 \neq \emptyset$. Hence, the undecidability follows.

In particular, Theorem 10 holds in the cases, where L and R are accepted by (1) deterministic pushdown automaton whose stack is 1-turn; (2) deterministic one-counter automata. Indeed, the disjointness problem is undecidable for the language family accepted by deterministic 1-turn pushdown automata (and deterministic one-counter automata, resp.), see, e.g, [I78]).

References

[BCCC96] L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi Reghizzi. Multiple pushdown languages and grammars. *Internat. J. Found. Comput. Sci.* **7** (1996), 253 – 291.

[ChK97] C. Choffrut and J. Karhumäki, Combinatorics of words, in Handbook of Formal Languages, Vol. 1, (A. Salomaa and G. Rozenberg, eds.), Springer-Verlag, 1997, pp. 329 – 438.

[ChK99] C. Choffrut and J. Karhumäki. Characterizing the subsets of words commuting with a set of two words. *Proc. 7th Nordic Combinatorial Conference*, Turku, Finland, 1999.

[ChKO99] C. Choffrut, J. Karhumäki, and N. Ollinger. The commutation of finite sets: challenging problem. *Theoret. Comput. Sci.*, to appear; TUCS Technical Report 303, http://www.tucs.fi, 1999.

[CS78] K. Culik II and A. Salomaa. On the decidability of homomorphism equivalence for languages. *J. Comput. System Sci.* **17**, (1978), 163 – 175.

[D00] Z. Dang. Verification and Debugging of Infinite State Real-time Systems. *Ph.D. Thesis*, University of California, Santa Barbara, 2000.

[Gi66] S. Ginsburg. The Mathematical Theory of Context-Free Languages. McGraw-Hill, New York, 1966.

[Gr68] S. A. Greibach. Checking automata and one-way stack languages. *SDC Document TM 738/045/00*, 1968.

[GuI81] E. M. Gurari and O. H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *J. Comput. System Sci.* **22** (1981), 220 – 229.

[HalH98] V. Halava and T. Harju. Undecidability in integer weighted finite automata. *Fund. Inf.* **38** (1999), 189 – 200.

[Har78] M. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, Mass., 1978.

[HoKK87] S. Horvath, J. Karhumäki, and H. C. M. Kleijn. Results concerning palindromicity. *J. Int. Process. Cyber. EIK* **23** (1987), 441 – 451.

[I78] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM* **25** (1978), 116 – 133.

[IBS00] O. H. Ibarra, T. Bultan, and J. Su. Reachability analysis for some models of infinite-state transition systems. *Proc. 10th Int. Conf. on Concurrency Theory*, 2000.

[IJTW95] O. H. Ibarra, T. Jiang, N. Tran, and H. Wang. New decidability results concerning two-way counter machines. *SIAM J. Comput.* **24** (1995), 123 – 137.

[KL00] J. Karhumäki and L. P. Lisovik. A simple undecidable problem: the inclusion problem for finite substitutions on ab^*c. *Proc. of the STACS 2001*, to appear.

[L83] M. Lothaire, *Combinatorics on Words*, Addison-Wesley, 1983.

[P66] R. Parikh. On context-free languages. *J. ACM* **13** (1966), 570 – 581.

[Sa73] A. Salomaa. *Formal Languages.* Academic Press, New York, 1973.

[Se97] G. Sénizergues. L(A)=L(B)? decidability results from complete formal systems. *Theoret. Comput. Sci.* **251** (2001), 1 – 166.

[St00] C. Stirling. Decidability of DPDA equivalence. *Theoret. Comput. Sci.*, to appear.

The Star Problem in Trace Monoids: Reductions Beyond C4*

(Extended Abstract)**

Daniel Kirsten

Institute of Algebra, Dresden University of Technology, D-01062 Dresden, Germany
kirsten@math.tu-dresden.de, www.math.tu-dresden.de/~kirsten

Abstract. We deal with the star problem in trace monoids which means to decide whether the iteration of a recognizable trace language is recognizable. We consider trace monoids $\mathbb{K}_n = \{a_1, b_1\}^* \times \cdots \times \{a_n, b_n\}^*$. Our main theorem asserts that the star problem is decidable in a trace monoid $\mathbb{M}$ iff it is decidable in the biggest $\mathbb{K}_n$ submonoid in $\mathbb{M}$. Thus, future research on the star problem can focus on the trace monoids $\mathbb{K}_n$. The recently shown decidability equivalence between the star problem and the finite power problem [14] plays a crucial role in the paper.

1 Introduction

We deal with the star problem in free, partially commutative monoids, also known as trace monoids. The star problem was raised by E. Ochmański in 1984 [20]. It means to decide whether the iteration of a recognizable trace language is recognizable.

Here, we show a new partial result: We deal with trace monoids of the form $\mathbb{K}_n = \{a_1, b_1\}^* \times \cdots \times \{a_n, b_n\}^*$. We show that the decidability of the star problem in some $\mathbb{K}_n$ implies its decidability in any other trace monoid without a $\mathbb{K}_{n+1}$ submonoid. Thus, future research on the star problem can focus on the trace monoids $\mathbb{K}_n$. Our main result includes Richomme's theorem which asserts the decidability of the star problem in trace monoids without a C4.

The paper is organized as follows: In Section 2, we explain basic notations and recall trace monoids, reognizable sets, and related notions. Section 3 is devoted to the star problem and the finite power problem. After a historical survey in Section 3.1 we explain the main results and interactions with previously known results in Section 3.2. In Section 3.3, we try to evaluate our contribution and to point out open questions which could be next research steps.

To keep Section 3 as a lucid survey, we give the main proof in Section 4: From Section 4.1 to 4.2, we reduce the star problem from some trace monoid $\mathbb{M}(A, D)$ to some trace monoid $\mathbb{M}(A, D')$, where (A, D) and (A, D') differ in

* Partially supported by the PhD program "Specification of discrete processes and systems of processes by operational models and logics" of the DFG.

** See author's homepage for a long version including complete proofs [13].

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 591–602, 2001.

just one dependency. In Section 4.3, we apply this step inductively to reduce the star problem in an arbitrary trace monoid to a trace monoid over a transitive dependency relation.

2 Preliminaries

2.1 Generalities

Let $\mathbb{N} = \{0, 1, 2, \dots\}$. For finite sets L, we denote by $|L|$ the number of elements of L. If $p \in L$, then we denote by p both the element p and the singleton set consisting of p. We denote by the symbols $\subseteq$ and $\subset$ set inclusion and strict set inclusion, respectively. For a binary relation D, we denote by $\mathsf{tr}D$ the transitive closure of D. For a mapping f, we denote by f^{-1} the inverse of f. We denote the power set of some set A by 2^A. We regard 2^A as a monoid by considering set union as the natural operation.

If $\mathbb{M}_1$ and $\mathbb{M}_2$ are two monoids, we define their *cartesian product* $\mathbb{M}_1 \times \mathbb{M}_2$ as the cartesian product of the underlying sets of $\mathbb{M}_1$ and $\mathbb{M}_2$ equipped with the componentwise operation of $\mathbb{M}_1$ and $\mathbb{M}_2$. We extend this definition in a natural way to more than two monoids.

For $L \subseteq \mathbb{M}$ and $n \leq m \in \mathbb{N}$, we denote $L^{n,\dots,m} = \bigcup_{i \in \{n,\dots,m\}} L^i$.

Let L be a subset of some monoid $\mathbb{M}$. The set L has the *finite power property* (for short FPP) iff there is some $n \in \mathbb{N}$ such that $L^* = L^{0,\dots,n}$. Let $\mathbb{M}'$ be a monoid and let $h : \mathbb{M} \to \mathbb{M}'$ be a homomorphism. If L has the FPP, then $h(L)$ has the FPP. If h is injective, then L has the FPP iff $h(L)$ has the FPP.

2.2 Free Monoids and Trace Monoids

We recall well-known basic notions. We denote the *free monoid* over an alphabet A by A^*. Its identity is the empty word ε. We define the length of words as the homomorphism $|| : A^* \to (\mathbb{N}, +)$ which is uniquely defined by $|a| = 1$ for $a \in A$. For every $b \in A$, we define the homomorhism $||_b : A^* \to (\mathbb{N}, +)$ by $|b|_b = 1$ and $|a|_b = 0$ for $a \neq b$. We define the alphabet of a word as the unique homomorphism $\alpha : A^* \to 2^A$ with $\alpha(a) = \{a\}$ for $a \in A$.

P. Cartier and D. Foata introduced the concept of the free partially commutative monoids, usually called trace monoids, in 1969 [2]. In 1977, A. Mazurkiewicz considered this concept as a potential model for concurrent systems [15]. Since then, trace monoids have been examined by both mathematicians and theoretical computer scientists [5,6]. Recently, results and techniques from trace theory turned out to be useful in the framework of message sequence charts.

Let A be an alphabet. We call a binary relation D over A a *dependence relation* iff D is reflexive and symmetric. For every pair of letters $a, b \in A$ with aDb, we say that a and b are *dependent*, otherwise a and b are *independent*. We call the pair (A, D) a *dependence alphabet*. We define the relation $\sim_D$ over A^* as the congruence induced by $ab \sim_D ba$ for independent letters $a, b \in A$.

We call the congruence classes of $\sim_D$ *traces*. We denote the factorization $A^*_{/\sim_D}$ by $\mathbb{M}(A, D)$ and call it the *trace monoid* over (A, D). We call subsets

of $\mathbb{M}(A, D)$ *trace languages*, or for short *languages*. We denote by $[\,]_D$, or for short $[\,]$, the canonical homomorphism from A^* to $\mathbb{M}(A, D)$. We denote by $[\,]_D^{-1}$ or $[\,]^{-1}$ its inverse.

Let $u, v \in A^*$ with $u \sim_D v$, i.e., $[u]_D = [v]_D$. We have $|u| = |v|$, $\alpha(u) = \alpha(v)$, and $|u|_a = |v|_a$ for $a \in A$. Hence, we can define these three homomorphisms for trace monoids. We call two traces s, t *independent* iff $\big(\alpha(s) \times \alpha(t)\big) \cap D = \emptyset$.

Let (A, D) be a dependence alphabet and let $B \subseteq A$. We denote the dependence alphabet $\big(B, D \cap (B \times B)\big)$ by (B, D).

If $D = A \times A$, then $\mathbb{M}(A, D)$ is (isomorphic to) the free monoid A^*. Cartesian products of trace monoids are (isomorphic to) trace monoids.

Let (A, D) be a dependence alphabet with a transitive dependence relation D, i.e., let D be an equivalence relation. The trace monoid $\mathbb{M}(A, D)$ is isomorphic to $A_1^* \times A_2^* \times \cdots \times A_n^*$, where $A_1, \ldots, A_n$ are the equivalence classes of D.

For $n > 0$, we define $\mathbb{K}_n = \{a_1, b_1\}^* \times \{a_2, b_2\}^* \times \cdots \times \{a_n, b_n\}^*$. We define $\mathbb{K}_0$ as the trivial monoid. The trace monoid $\mathbb{K}_2$ is well-known in trace theory as C4. Further, the trace monoid $\{a_1, b_1\}^* \times \{a_2\}^*$ is usually called P3.

The notion of *connectivity* plays a key role in trace theory, particularly in the research on the star problem. We call a dependence alphabet (A, D) *connected* iff we cannot split A into two non-empty, mutually disjoint subsets A_1 and A_2 with $(A_1 \times A_2) \cap D = \emptyset$. We call a trace $t \in \mathbb{M}(A, D)$ *connected* iff the dependence alphabet $\big(\alpha(t),\ D \cap \alpha(t) \times \alpha(t)\big)$ is connected. We call a trace language L *connected* iff every trace in L is connected.

G. Pighizzini introduced the notion of *restrictions* [22]: Let (A, D) be a dependence alphabet and let $B \subseteq A$. For a language $L \subseteq \mathbb{M}(A, D)$, we define

$$L_{\subseteq B} = \big\{\, t \in L \,\big|\, \alpha(t) \subseteq B \,\big\} \qquad L_{=B} = \big\{\, t \in L \,\big|\, \alpha(t) = B \,\big\}$$

We abbreviate $(L^*)_{\subseteq B} = (L_{\subseteq B})^*$ by $L^*_{\subseteq B}$. However, $(L^*)_{=B}$ and $(L_{=B})^*$ are not necessarily equal, e.g., if $B = L = A$ and $|A| \geq 2$, then $(A^*)_{=A} \neq (A_{=A})^* = \emptyset^*$.

2.3 Recognizable Languages

We recall some notions and results concerning recognizable sets. See [1] for more information. Let $\mathbb{M}$ be a monoid. We call a triple $\mathfrak{A} = [Q, h, F]$ consisting of a finite monoid Q, a homomorphism $h : \mathbb{M} \to Q$ and a subset $F \subseteq Q$ an $\mathbb{M}$-*automaton*, or simply *automaton*. We call the set $h^{-1}(F)$ the language (or set) of the automaton $\mathfrak{A}$ and denote it by $L(\mathfrak{A})$. We call some subset $L \subseteq \mathbb{M}$ *recognizable* over $\mathbb{M}$ iff there exists an $\mathbb{M}$-automaton $\mathfrak{A}$ such that $L = L(\mathfrak{A})$. Then, we also say that $\mathfrak{A}$ recognizes L or $\mathfrak{A}$ is an automaton for L. We denote the class of all recognizable sets over $\mathbb{M}$ by $\mathrm{REC}(\mathbb{M})$.

In any monoid $\mathbb{M}$, the empty set and $\mathbb{M}$ itself are recognizable. The family $\mathrm{REC}(\mathbb{M})$ is closed under union, intersection, and complement. Moreover, if $\mathbb{M}$ and $\mathbb{M}'$ are two monoids and $g : \mathbb{M} \to \mathbb{M}'$ is a homomorphism, then we have the following properties:

1. For every set $L' \in \mathrm{REC}(\mathbb{M}')$, we have $g^{-1}(L') \in \mathrm{REC}(\mathbb{M})$.
2. If g is surjective and $L' \subseteq \mathbb{M}'$, then $g^{-1}(L') \in \mathrm{REC}(\mathbb{M})$ iff $L' \in \mathrm{REC}(\mathbb{M}')$.

The study of recognizable trace languages is a central part in trace theory [5,19]. Every finite trace language is recognizable. Since the homomorphism [ ] is surjective, a trace language L is recognizable iff $[L]^{-1}$ is recognizable. M. Fliess, R. Cori, and D. Perrin showed that the concatenation of two recognizable trace languages yields a recognizable trace language [4,8]. Recognizable trace languages are not closed under iteration. Consider the trace monoid $a^* \times b^*$. The iteration of the singleton language $\{(a,b)\}$ yields $\{\,(a^n, b^n) \mid n \in \mathbb{N}\,\}$ which is not recognizable, because its inverse image $\left\{\, w \;\middle|\; |w|_a = |w|_b \right\} \subseteq \{a,b\}^*$ is not recognizable. However, the iteration of a connected recognizable trace language always yields a recognizable trace language [3,16,20].

Let $\mathbb{M}$ be a trace monoid. E. Ochmański showed that the class $\mathrm{REC}(\mathbb{M})$ is the least class which contains every finite subset of $\mathbb{M}$ and is closed under union, concatenation, and iteration of connected languages [19,20].

The restrictions $L_{=B}$ and $L_{\subseteq B}$ preserve recognizability [14,22].

Note that every closure property which we mention is effective, i.e., we can construct automata for the desired sets.

3 The Star Problem and the FPP

3.1 A Historical Survey

The *star problem* was raised by E. Ochmański in 1984 [20]. It means to decide *whether* for a recognizable trace language L, the iteration L^* is recognizable. Recently, it turned out that the star problem for message sequence charts is equivalent to the star problem for trace monoids [18].

The *finite power problem* means to decide *whether* a recognizable trace language L has the *finite power property*, i.e., it means to decide whether there is some $n \in \mathbb{N}$ such that $L^* = L^{0,\cdots,n}$. It was raised for the first time just for free monoids by J. A. Brzozowski in 1966. In 1990, E. Ochmański considered the finite power problem in trace monoids [21]. We abbreviate both the terms *finite power problem* and *finite power property* by FPP.

We say that the star problem (resp. FPP) is decidable in some trace monoid $\mathbb{M}(A,D)$ iff it is decidable for recognizable languages over $\mathbb{M}(A,D)$. To say that the star problem (resp. FPP) is decidable for a class of recognizable languages over $\mathbb{M}(A,D)$ with some certain property, we say, e.g., that it is decidable for $L \in \mathrm{REC}(\mathbb{M}(A,D))$ with $L \subseteq \mathbb{M}(A,D)_{=A}$.

There is an algorithm which effectively constructs an automaton for L^* from an automaton for L [5]. This algorithm terminates iff L^* is recognizable.

Although during the recent 17 years many papers have dealt with the star problem and the FPP, only partial results have been achieved. In free monoids, the star problem is obvious and the FPP is decidable [11,25]. The star problem is decidable in free commutative monoids due to [10].

In the eighties, E. Ochmański [20], M. Clerbout and M. Latteux [3], and Y. Métivier [16] independently proved that the iteration of a connected recognizable trace language yields a recognizable trace language.

In 1992, J. Sakarovitch showed as a conclusion from a more general result the decidability of the star problem in trace monoids without a submonoid isomorphic to P3 [24]. In the same year, P. Gastin, E. Ochmański, A. Petit, and B. Rozoy proved the decidability of the star problem in P3 [9].

In 1994, Y. Métivier and G. Richomme showed the decidability of the FPP for connected recognizable trace languages and the decidability of the star problem for particular cases of finite trace languages [17].

In 1994, G. Richomme combined and improved various approaches and proved the following theorem [14,23]:

Theorem 1. *Let $\mathbb{M}$ be a trace monoid. If $\mathbb{M}$ does not contain a* C4 *submonoid, then both the star problem and the FPP are decidable in $\mathbb{M}$.*

He also showed the following reduction [14,23]:

Theorem 2. *Let (A, D) be a connected dependence alphabet. To show the decidability of the star problem in $\mathbb{M}(A, D)$, it suffices to show its decidability in $\mathbb{M}(B, D)$ for every strict subset $B \subset A$.*

The subsequent years were designated by stagnation. The star problem and the FPP were given up and one ceased the research.

In 1999, D. Kirsten showed a crucial connection between the star problem and the FPP [12,14]. In combination with earlier results due to G. Richomme, we proved the following theorem [14]:

Theorem 3. *The trace monoids with a decidable star problem are exactly the trace monoids with a decidable FPP.*

In [14], D. Kirsten and G. Richomme give a comprehensive presentation of an approach which is based on induction steps on dependence alphabets. In the present paper, we need two more results from [14]:

Theorem 4. *Let (A, D) be a dependence alphabet and let $b \notin A$ be a letter. If the star problem is decidable in $\mathbb{M}(A, D)$, then it is also decidable in $\mathbb{M}(A, D) \times b^*$.*

Note that by Theorem 3 the same assertion holds for the FPP. A weaker version of Theorem 4 occured in [23]. However, in [23], G. Richomme assumed that *both* the star problem *and* the FPP are decidable in $\mathbb{M}(A, D)$, because Theorem 3 was not known. The next theorem follows from results in [14]. See [13] for a proof.

Theorem 5. *Let (A, D) be a dependence alphabet. If the FPP is decidable for*

1. *$L \in \mathrm{REC}(\mathbb{M}(A, D))$ with $L \subseteq (\mathbb{M}(A, D))_{=A}$ and for*
2. *$L \in \mathrm{REC}(\mathbb{M}(B, D))$ for every strict subset $B \subset A$,*

then the FPP is decidable in $\mathbb{M}(A, D)$.

3.2 Main Results

At first, we reduce the FPP for a particular class of languages in arbitrary trace monoids to the FPP in trace monoids over transitive dependencies.

Theorem 6. *Let (A, D) be a dependence alphabet. If the FPP is decidable for $L \in \mathrm{REC}(\mathbb{M}(A, \mathsf{tr}D))$ with $L \subseteq \mathbb{M}(A, \mathsf{tr}D)_{=A}$, then the FPP is also decidable for $L \in \mathrm{REC}(\mathbb{M}(A, D))$ with $L \subseteq \mathbb{M}(A, D)_{=A}$.*

We will prove this result in Section 4. We show a proposition which allows to reduce the FPP from arbitrary cartesian products of free monoids to some $\mathbb{K}_n$.

Proposition 1. *Let (A, D) be a dependence alphabet, and let D be transitive. Let n be the number of non-singleton components of (A, D). If the FPP is decidable in $\mathbb{K}_n$, then the FPP is decidable in $\mathbb{M}(A, D)$.*

Proof. Note that $\mathbb{M}(A, D)$ is isomorphic to $A_1^* \times \cdots \times A_n^* \times b_1^* \times \cdots \times b_m^*$ where $A_1, \ldots, A_n$ are the non-singleton equivalence classes of D and $b_1, \ldots, b_m \in A$.

By Theorem 4, it suffices to show the decidability of the FPP in $A_1^* \times \cdots \times A_n^*$. There is an injective homomorphism $h : A_1^* \times \cdots \times A_n^* \to \mathbb{K}_n$ such that h maps connected traces to connected traces, i.e., h preserves both recognizability [7,19] and the FPP. □

From Theorem 6 and Proposition 1, we easily deduce our main result:

Theorem 7. *Let $n \geq 1$. If the star problem is decidable in $\mathbb{K}_n$, then the star problem is decidable in every trace monoid without a submonoid $\mathbb{K}_{n+1}$.*

Proof. By the equivalence between the star problem and the FPP (Theorem 3), we can prove Theorem 7 by showing the same reduction for the FPP. Let

$$\mathcal{C} = \big\{ (A, D) \,\big|\, \mathbb{M}(A, D) \text{ does not contain a } \mathbb{K}_{n+1} \text{ submonoid} \big\}.$$

We show by an induction on dependence alphabets that the FPP is decidable in $\mathbb{M}(A, D)$ for every $(A, D) \in \mathcal{C}$. Let $(A, D) \in \mathcal{C}$ be arbitrary.

If $|A| = 1$, then the FPP is decidable [11,25].

Let $|A| > 1$. Every strict subalphabet of (A, D) belongs to $\mathcal{C}$. By induction, the FPP is decidable in $\mathbb{M}(B, D)$ for every strict $B \subset A$. By Theorem 5, it suffices to show the decidability of the FPP for $L \in \mathrm{REC}(\mathbb{M}(A, D))$ with $L \subseteq \mathbb{M}(A, D)_{=A}$. By Theorem 6, it suffices to show the decidability of the FPP for $L \in \mathrm{REC}(\mathbb{M}(A, \mathsf{tr}D))$ with $L \subseteq \mathbb{M}(A, D)_{=A}$. By Proposition 1, it suffices to show the decidability of the FPP in $\mathbb{K}_k$, where k is the number of non-singleton components of $(A, \mathsf{tr}D)$, i.e., the number of non-singleton components of (A, D).

By choosing two dependent letters from each non-singleton component, we locate a $\mathbb{K}_k$ submonoid in $\mathbb{M}(A, D)$. There is no $\mathbb{K}_{n+1}$ submonoid in $\mathbb{M}(A, D)$. Hence, $k \leq n$, i.e., $\mathbb{K}_k$ is a submonoid of $\mathbb{K}_n$, and the FPP is decidable in $\mathbb{K}_k$. □

To illustrate Theorem 7, we state two corollaries:

Corollary 1. *To show the decidability of the star problem in a trace monoid $\mathbb{M}$, it suffices to show its decidability in the biggest $\mathbb{K}_n$ submonoid of $\mathbb{M}$.*

Corollary 2. *Exactly one of the following assertions is true:*

1. *The star problem is decidable in every trace monoid.*
2. *There is some $n > 1$ such that the trace monoids with a decidable star problem are exactly the trace monoids without a $\mathbb{K}_n$ submonoid.*

3.3 Conclusions and Open Problems

The main conclusion from Theorem 7 is that future research on the star problem and the FPP can focus on the monoids $\mathbb{K}_n$. Corollary 1 strictly subsumes the previously known reduction steps in Theorems 2 and 4. If we set $n = 1$ in Theorem 7, then we obtain Theorem 1 due to G. Richomme as a particular case of Theorem 7.

There is another well-known decision problem in trace theory: the code problem. Opposed to the star problem, there are trace monoids for which the code problem is known to be undecidable. For the code problem, the border between decidability and undecidability follows (as far as known) a rather mysterious way. By Corollary 2, we know that this is not the case for the star problem.

This paper gives an application of the recently shown decidability equivalence between the star problem and the FPP [14]. The encouraged reader is invited to prove Theorem 7 and an analogon of Theorem 6 without using Theorem 3.

Despite this progress, several interesting questions remain open. The decidability of the star problem in the monoids $\mathbb{K}_n$ and in particular in C4 is still not known. Further, we do not know whether the star problem in any trace monoid with a C4 submonoid is equivalent to the star problem in C4.

4 Reduction to Transitive Dependencies

To prove Theorem 6, we show the following slightly weaker proposition:

Proposition 2. *Let (A, D) be a dependence alphabet with letters $a, b, c \in A$ such that aDb, bDc, but $\neg\, aDc$. Let $D' = D \cup \{(a, c), (c, a)\}$. If the FPP is decidable for $L \in \mathrm{REC}(\mathbb{M}(A, D'))$ with $L \subseteq \mathbb{M}(A, D')_{=A}$, then the FPP is decidable for $L \in \mathrm{REC}(\mathbb{M}(A, D))$ with $L \subseteq \mathbb{M}(A, D)_{=A}$.*

On the one hand, the reduction from $\mathbb{M}(A, D)$ to $\mathbb{M}(A, D')$ is technically much easier than the reduction from $\mathbb{M}(A, D)$ to $\mathbb{M}(A, \mathsf{tr}D)$ in Theorem 6, because (A, D) and (A, D') differ in just one dependency. On the other hand, we will obtain Theorem 6 as an obvious conclusion from Proposition 2 in the last line of this section. We fix $\mathbb{M} = \mathbb{M}(A, D)$, $\mathbb{M}' = \mathbb{M}(A, D')$, and a, b, c from Proposition 2 for the rest of the paper.

4.1 A Little Bit of Transitivity

In this section, we show the following reduction:

Proposition 3. *If the FPP is decidable for $L \in \mathrm{REC}(\mathbb{M}')$ with $L \subseteq \mathbb{M}'_{=A}b$, then the FPP is decidable for $L \in \mathrm{REC}(\mathbb{M})$ with $L \subseteq \mathbb{M}_{=A}\, b$.*

We consider the canonical homomorphism $[\![]\!] : \mathbb{M}' \to \mathbb{M}$ which is induced by the identity on letters. For a language $L \subseteq \mathbb{M}'$, the equation $L = [\![[\![L]\!]]\!]^{-1}$ means that L is closed under commutation of a and c. We need a rather technical lemma:

Lemma 1. *Let $K \subseteq \mathbb{M}'b$, $L \subseteq \mathbb{M}'$ with $K = [\![[\![K]\!]]\!]^{-1}$ and $L = [\![[\![L]\!]]\!]^{-1}$. We have $KL = [\![[\![KL]\!]]\!]^{-1}$, $K^{0,\ldots,k} = [\![[\![K^{0,\ldots,k}]\!]]\!]^{-1}$ (for $k \in \mathbb{N}$), and $K^* = [\![[\![K^*]\!]]\!]^{-1}$.*

Proof. (sketch) We show $KL = [\![[\![KL]\!]]^{-1}$, i.e., we show that KL is closed under commutation of a's and c's. Let $t \in KL$. There are $rb \in K$ and $s \in L$ with $rbs = t$. If we transform t into some trace t' by commuting consecutive a's and c's, then we cannot commute beyond the b between r and s, i.e, we commute inside r and inside s. Hence, we have $t' = r'bs'$ where we obtained r' and s' from r and s, resp., by commuting consecutive a's and c's. Thus, we have $r'b \in K$, $s' \in L$, and $r'bs' \in KL$.

We can show the other assertions by a straightforward induction on k and considering that $K^* = \bigcup_{k \in \mathbb{N}} K^k$.

See [13] for a formal precise proof based on Levi's Lemma. □

Example 1. Let $K = \{a\}$ and $L = \{c\}$. We have $a = [\![[\![a]\!]]^{-1}$ and $c = [\![[\![c]\!]]^{-1}$, but $KL = \{ac\} \neq \{ac, ca\} = [\![[\![KL]\!]]^{-1}$. Consequently, the assumption that every trace in K has a trailing b must not be dropped.

Lemma 2. *Let $L \subseteq \mathbb{M}\,b$. The language L has the FPP iff $[\![L]\!]^{-1}$ has the FPP.*

Proof. At first, note that $L = [\![[\![L]\!]^{-1}]\!]$.

$\ldots \Rightarrow \ldots$ Choose some $k \in \mathbb{N}$ with $L^* = L^{0,\ldots,k}$. By $L = [\![[\![L]\!]^{-1}]\!]$, we have $[\![[\![L]\!]^{-1}]\!]^* = [\![[\![L]\!]^{-1}]\!]^{0,\ldots,k}$. We obtain $[\![\left([\![L]\!]^{-1}\right)^*]\!] = [\![\left([\![L]\!]^{-1}\right)^{0,\ldots,k}]\!]$ because $[\![\,]\!]$ is a homomorphism. We apply $[\![\,]\!]^{-1}$ to both sides and use Lemma 1 with $[\![L]\!]^{-1}$ as K. We obtain $\left([\![L]\!]^{-1}\right)^* = \left([\![L]\!]^{-1}\right)^{0,\ldots,k}$, i.e., $[\![L]\!]^{-1}$ has the FPP.

$\ldots \Leftarrow \ldots$ Homomorphisms preserve the FPP, i.e., $[\![[\![L]\!]^{-1}]\!] = L$ has the FPP. □

Proof of Proposition 3. To decide whether $L \in \mathrm{REC}(\mathbb{M})$ has the FPP it suffices by Lemma 2 to decide whether $[\![L]\!]^{-1} \in \mathrm{REC}(\mathbb{M}')$ has the FPP. □

4.2 On Trailing *b*'s

This section is devoted to the proof of the following reduction, where we still assume $\mathbb{M}$ and a, b, c from the beginning of Section 4.

Proposition 4. *If the FPP is decidable for $L \in \mathrm{REC}(\mathbb{M})$ with $L \subseteq \mathbb{M}_{=A}b$, then the FPP is also decidable for languages $L \in \mathrm{REC}(\mathbb{M})$ with $L \subseteq \mathbb{M}_{=A}b\mathbb{M}_{=A}$.*

We transform a given recognizable language $L \subseteq \mathbb{M}_{=A}b\mathbb{M}_{=A}$ into a recognizable language $L' \subseteq \mathbb{M}_{=A}b$ such that L' has the FPP iff L has the FPP.

Let $L \subseteq \mathbb{M}_{=A}b\mathbb{M}_{=A}$ be a language, and let $\mathfrak{A} = [Q, h, F]$ be an automaton with $L = L(\mathfrak{A})$ for the rest of this section. We can freely assume that h is surjective and that for $s, t \in \mathbb{M}$, $h(s) = h(t)$ implies $\alpha(s) = \alpha(t)$. Consequently, $\alpha : Q \to 2^A$ with $\alpha(q) = \alpha(t)$ for every $q \in Q$ and $t \in h^{-1}(q)$ is a homomorphism.

We consider the submonoid of $\mathbb{M}$ which is generated by $(A \setminus b) \cup b^{|Q|+1}$. The unique homomorphism $g : \mathbb{M} \to \left[(A \setminus b) \cup b^{|Q|+1}\right]^*_D$ induced by

$$g(a) = a \quad \text{for } a \in A \setminus b, \qquad \text{and} \quad g(b) = b^{|Q|+1}$$

is an isomorphism. We can regard g as an injective embedding of $\mathbb{M}$ into itself. Moreover, g maps connected traces to connected traces, because $\alpha(t) = \alpha(g(t))$ for $t \in \mathbb{M}$. Hence, g preserves recognizability [7,19].

Lemma 3. *The language $L \subseteq \mathbb{M}$ has the FPP iff $g(L)$ has the FPP.*

Proof. As mentioned above, g is injective. □

In the rest of this section, we deal with $g(L)$. Let $\# : Q \to \{1, \ldots, |Q|\}$ be some injective mapping. For $q \in Q$, we denote $\#(q)$ by $\#q$.

We define three languages L_1, L_2, and T. Later, the language $(L_1 L_2 \cup bTb)$ plays the role of the language L' which we mentioned above. We define:

$$L_1 = \bigcup_{m,p \in Q,\ \alpha(p) = A,\ m\,h(b)\,p \in F} b^{|Q|+1-\#m} g(h^{-1}(p))$$

$$L_2 = \bigcup_{q \in Q,\ \alpha(q) = A} g(h^{-1}(q)) b^{\#q}$$

Note that $L_1 \subseteq b\,\mathbb{M}_{=A} \subseteq \mathbb{M}_{=A}$ and $L_2 \subseteq \mathbb{M}_{=A} b \subseteq \mathbb{M}_{=A}$. We can construct automata for L_1 and L_2. Let $\pi_{a,b} : \mathbb{M} \to \{a, b\}^*$ be the projection, and let:

$$T = \pi_{a,b}^{-1}\Big(\{a,b\}^* a \big\{b^{|Q|+1}\big\}^* b^{1,\ldots,|Q|} a \{a,b\}^*\Big) \cap \mathbb{M}_{=A},$$

i.e., a trace $t \in \mathbb{M}_{=A}$ belongs to T iff $\pi_{a,b}(t)$ contains some subword ab^+a whose number of b's is not a multiple of $|Q| + 1$.

Lemma 4. *We have*

1. $g(\mathbb{M}) \cap T = \emptyset$, *(and hence, $g(L) \cap T = \emptyset$),*
2. $L_2 L_1 \subseteq g(L) \cup T$,
3. $L_2 L_1 \setminus T = g(L)$, *and*
4. $\mathbb{M}\,T\,\mathbb{M} = T$.

Proof. (1) Note that $\pi_{a,b}(g(\mathbb{M}))$ is the free monoid over $\{a, b^{|Q|+1}\}$ and consider the definition of T.

(2) Let $s \in L_2$ and $t \in L_1$. We have $st \in \mathbb{M}_{=A}$. By the definition of L_1 and L_2, there are q, m, $p \in Q$ with $\alpha(p) = \alpha(q) = A$, $mh(b)p \in F$ such that $s \in g(h^{-1}(q))b^{\#q}$ and $t \in b^{|Q|+1-\#m} g(h^{-1}(p))$.

Case a: Let $m = q$. We show $st \in g(L)$. By $m = q$, we have

$$\begin{aligned} st &\in g(h^{-1}(m))\, b^{\#m}\, b^{|Q|+1-\#m}\, g(h^{-1}(p)) \\ &= g\big(h^{-1}(m)\big)\, g\big(b\big)\, g\big(h^{-1}(p)\big) \\ &= g\big(h^{-1}(m)\, b\, h^{-1}(p)\big) \\ &\subseteq g(L) \end{aligned}$$

The latter inclusion follows from $m\,h(b)\,p \in F$.

Case b: Let $m \neq q$. We show $st \in T$. We have

$$st \in g(h^{-1}(q))\, b^{\#q+|Q|+1-\#m}\, g(h^{-1}(p))$$
$$\subseteq g(\mathbb{M}_{=A})\, b^{|Q|+1+\#q-\#m}\, g(\mathbb{M}_{=A})$$

Every trace in $g(\mathbb{M}_{=A})$ contains every letter of A, and thus, in particular the letter a. Hence, we have
$\pi_{a,b}(g(\mathbb{M}_{=A})) \subseteq \{a, b^{|Q|+1}\}^* a \{b^{|Q|+1}\}^* = \{b^{|Q|+1}\}^* a \{a, b^{|Q|+1}\}^*$.
Now, we consider $\pi_{a,b}(st)$.

$$\pi_{a,b}(st) \in \pi_{a,b}(g(\mathbb{M}_{=A}))\, \pi_{a,b}(b^{|Q|+1+\#q-\#m})\, \pi_{a,b}(g(\mathbb{M}_{=A}))$$
$$\subseteq \{a, b^{|Q|+1}\}^* a \{b^{|Q|+1}\}^*\, b^{|Q|+1+\#q-\#m}\, \{b^{|Q|+1}\}^* a \{a, b^{|Q|+1}\}^*$$
$$\subseteq \{a, b\}^* a \{b^{|Q|+1}\}^*\, b^{|Q|+1+\#q-\#m}\, a \{a, b\}^*$$

Then, $st \in T$ follows from $\#m \neq \#q$ and $st \in \mathbb{M}_{=A}$.

(3) By (2) and (1), it suffices to show $g(L) \subseteq L_2 L_1$. Let $t \in L$. We show $g(t) \in L_2 L_1$. Because $L \subseteq \mathbb{M}_{=A} b \mathbb{M}_{=A}$, we have $t = rbs$ for some $r, s \in \mathbb{M}_{=A}$. Let $q = h(r)$ and $p = h(s)$. We have $\alpha(q) = \alpha(p) = A$. By $t \in L$, we have $q\, h(b)\, p \in F$. We have

$$g(t) = g(rbs) \in g\big(h^{-1}(q)\big)\, b^{\#q}\, b^{|Q|+1-\#q}\, g\big(h^{-1}(p)\big) \subseteq L_2 L_1.$$

(4) Note that $\pi_{a,b}(\mathbb{M}) = \{a, b\}^*$ and consider the definition of T. We easily obtain $\pi_{a,b}(\mathbb{M}\, T\, \mathbb{M}) = \pi_{a,b}(T)$. □

The following lemma shows the connection between $L_1 L_2 \cup bTb$ and $g(L)$.

Lemma 5. *The language $g(L)$ has the FPP iff $(L_1 L_2 \cup bTb)$ has the FPP.*

Proof. ... $\Leftarrow$ ... Let $k \in \mathbb{N}$ with $\big(L_1 L_2 \cup bTb\big)^* = \big(L_1 L_2 \cup bTb\big)^{0,\ldots,k}$. Let $n > 1$ be arbitrary

$$\begin{aligned} g(L)^n &= (L_2 L_1 \setminus T)^n && \text{(cf. Lemma 4 (3))} \\ &\subseteq (L_2 L_1)^n \\ &= L_2 (L_1 L_2)^{n-1} L_1 \\ &\subseteq L_2 (L_1 L_2 \cup bTb)^{n-1} L_1 \\ &\subseteq L_2 (L_1 L_2 \cup bTb)^{0,\ldots,k} L_1 && \text{(cf. Lemma 4 (4))} \\ &\subseteq L_2 (L_1 L_2)^{0,\ldots,k} L_1 \cup T \end{aligned}$$

We have $g(L)^n = g(L^n) \subseteq g(\mathbb{M})$. By Lemma 4 (1), $g(L)^n \cap T = \emptyset$. Consequently,

$$\begin{aligned} g(L)^n &\subseteq L_2 (L_1 L_2)^{0,\ldots,k} L_1 \\ &= (L_2 L_1)^{1,\ldots,k+1} && \text{(cf. Lemma 4 (2))} \\ &\subseteq (g(L) \cup T)^{1,\ldots,k+1} && \text{(cf. Lemma 4 (4))} \\ &= g(L)^{1,\ldots,k+1} \cup T \end{aligned}$$

Once again, we apply $g(L)^n \cap T = \emptyset$, and obtain $g(L)^n \subseteq g(L)^{1,\ldots,k+1}$. Because $n > 1$ is chosen arbitrarily, we have $g(L)^* = g(L)^{0,\ldots,k+1}$, so $g(L)$ has the FPP.

... $\Rightarrow$ ... Let $k \in \mathbb{N}$ with $g(L)^* = g(L)^{0,\ldots,k}$. At first, note that

$$\big(L_1 L_2 \cup bTb\big)^*\, bTb\, \big(L_1 L_2 \cup bTb\big)^* \subseteq bTb$$

as a conclusion from Lemma 4 (4). To show that $L_1 L_2 \cup bTb$ has the FPP, it suffices to show $(L_1 L_2)^n \subseteq (L_1 L_2 \cup bTb)^{1,\ldots,k+1}$ for every $n > 1$. We have

$$
\begin{aligned}
(L_1L_2)^n &= L_1(L_2L_1)^{n-1}L_2 && \text{(cf. Lemma 4 (2))}\\
&\subseteq L_1\big(g(L)\cup T\big)^{n-1}L_2 && \text{(cf. Lemma 4 (4))}\\
&\subseteq L_1\big(g(L)\big)^{n-1}L_2 \cup bTb\\
&\subseteq L_1\big(g(L)\big)^{1,\ldots,k}L_2 \cup bTb && \text{(cf. Lemma 4 (3))}\\
&\subseteq L_1\big(L_2L_1\big)^{1,\ldots,k}L_2 \cup bTb\\
&\subseteq (L_1L_2)^{2,\ldots,k+1} \cup bTb\\
&\subseteq (L_1L_2 \cup bTb)^{1,\ldots,k+1}
\end{aligned}
$$

□

Proof of Proposition 4. Let $L = L(\mathfrak{A})$. To determine whether L has the FPP, we construct some automaton $\mathfrak{A}'$ with $L(\mathfrak{A}') = (L_1L_2 \cup bTb)$. By Lemmas 3 and 5, L has the FPP iff $L_1L_2 \cup bTb$ has the FPP. We can decide the latter condition, because $(L_1L_2 \cup bTb) \subseteq \mathbb{M}_{=A}b$. □

4.3 Completion of the Proof

At first, we show the following reduction:

Proposition 5. *Let $\mathbb{M}$ be the trace monoid over a dependence alphabet (A, D) and let $b \in A$. If the FPP is decidable for $L \in \mathrm{REC}(\mathbb{M})$ with $L \subseteq \mathbb{M}_{=A}b\,\mathbb{M}_{=A}$, then the FPP is decidable for $L \in \mathrm{REC}(\mathbb{M})$ with $L \subseteq \mathbb{M}_{=A}$.*

Proof. Let $L \in \mathrm{REC}(\mathbb{M})$ with $L \subseteq \mathbb{M}_{=A}$. We have $L^{3,\ldots,5} \in \mathrm{REC}(\mathbb{M})$ and $L^{3,\ldots,5} \subseteq \mathbb{M}_{=A}b\,\mathbb{M}_{=A}$. Further, L has the FPP iff $L^{3,\ldots,5}$ has the FPP. □

Proof of Proposition 2. We concatenate Propositions 3, 4, and 5. □

Proof of Theorem 6. We apply Proposition 2 inductively. □

Acknowledgements. A question from V. Diekert at the workshop "Logic and Algebra in Concurrency" initiated some research which lead to the present paper. The author thanks M. Droste, P. Gastin, M. Lohrey, G. Richomme, and anonymous referees for reading preliminary versions and giving useful remarks.

References

1. J. Berstel. *Transductions and Context-Free Languages.* B.G.Teubner, Stutt., 1979.
2. P. Cartier and D. Foata. *Problèmes combinatoires de commutation et réarrangements*, vol. 85 of *Lecture Notes in Mathematics.* Springer-Verlag, Berlin, 1969.
3. M. Clerbout and M. Latteux. Semi-commutations. *Inf. and Comp.*, 73:59–74, 1987.
4. R. Cori and D. Perrin. Automates et commutations partielles. *R.A.I.R.O. - Informatique Théorique et Applications*, 19:21–32, 1985.
5. V. Diekert and Y. Métivier. Partial commutation and traces. In G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages, Vol. 3, Beyond Words*, pages 457–534. Springer-Verlag, Berlin, 1997.
6. V. Diekert and G. Rozenberg, eds., *The Book of Traces.* World Scient., 1995.
7. C. Duboc. Mixed product and asynchronous automata. *Theoretical Computer Science*, 48:183–199, 1986.

8. M. Fliess. Matrices de Hankel. *J. de Math. Pures et Appl.*, 53:197–224, 1974.
9. P. Gastin, E. Ochmański, A. Petit, and B. Rozoy. Decidability of the star problem in $A^* \times \{b\}^*$. *Information Processing Letters*, 44(2):65–71, 1992.
10. S. Ginsburg and E. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.
11. K. Hashiguchi. A decision procedure for the order of regular events. *Theoretical Computer Science*, 8:69–72, 1979.
12. D. Kirsten. A connection between the star problem and the finite power property in trace monoids. In P. van Emde Boas et al., eds., *ICALP'99 Proceedings*, vol. 1644 of *LNCS*, pages 473–482. Springer-Verlag, Berlin, 1999.
13. D. Kirsten. The star problem in trace monoids: Reductions beyond C4. Technical Report MATH-AL-01-2001, Dresden University of Technology, 2001. (submitted)
14. D. Kirsten and G. Richomme. Decidability equivalence between the star problem and the finite power problem in trace monoids. *Theory of Computing Systems*, 34:3:193–227, 2001.
15. A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, 1977.
16. Y. Métivier. Une condition suffisante de reconnaissabilité dans un monoïde partiellement commutatif. *R.A.I.R.O. - Inform. Théor. et Appl.*, 20:121–127, 1986.
17. Y. Métivier and G. Richomme. New results on the star problem in trace monoids. *Information and Computation*, 119(2):240–251, 1995.
18. R. Morin. On regular MSC languages and relationships to Mazurkiewicz trace theory. In F. Honsell and M. Miculan, eds., *FoSSaCS'2001 Proceedings*, vol. 2030 of *LNCS*, pages 332–346. Springer-Verlag, Berlin, 2001.
19. E. Ochmański. Recognizable trace languages. Chapter 6 in [6], pages 167-204.
20. E. Ochmański. *Regular Trace Languages (in Polish).* PhD thesis, Warszawa, 1984.
21. E. Ochmański. Notes on a star mystery. *Bulletin of the EATCS*, 40:252–257, 1990.
22. G. Pighizzini. Synthesis of nondeterministic asynchronous automata. In M. Droste and Y. Gurevich, eds., *Semantics of Progr. Lang. and Model Theory*, number 5 in Algebra, Logic and Appl., p. 109–126. Gordon and Breach Sc. Publ., 1993.
23. G. Richomme. Some trace monoids where both the star problem and the finite power property problem are decidable. In I. Privara et al., eds., *MFCS'94 Proceedings*, vol. 841 of *LNCS*, pages 577–586. Springer-Verlag, Berlin, 1994.
24. J. Sakarovitch. The "last" decision problem for rational trace languages. In I. Simon, ed., *LATIN'92 Proc.*, vol. 583 of *LNCS*, p. 460–473. Springer-Verlag, 1992.
25. I. Simon. Limited subsets of a free monoid. In *Proceedings of the 19th IEEE Annual Symposium on Found. of Comp. Sc.*, pages 143–150. North Carolina Press, 1978.

The Trace Coding Problem Is Undecidable
(Extended Abstract)*

Michal Kunc

Department of Mathematics, Masaryk University,
Janáčkovo nám. 2a, 662 95 Brno, Czech Republic,
kunc@math.muni.cz, http://www.math.muni.cz/~kunc

Abstract. We introduce the notion of weak morphisms of trace monoids and use it to deal with the problem of deciding the existence of codings between trace monoids. We prove that this problem is not recursively enumerable, which answers the question raised by Ochmański in 1988. We also show its decidability when restricted to instances with domain monoids defined by acyclic dependence graphs.

1 Introduction and Notations

In [9] Mazurkiewicz proposed trace monoids as a tool for describing a behaviour of concurrent systems. When we consider simulations of these systems, trace morphisms come into play. In this context, the notion of a uniquely decipherable morphism (a coding) turns out to be particularly interesting. In [10] Ochmański raised a problem to find an algorithm deciding for each pair of trace monoids whether there is a coding between them. Up to now, only few partial decidability results concerning this problem were obtained in [1,2,6]. The problem was solved completely just for so-called strong morphisms by Diekert et al. [6].

In the present paper we introduce a new concept of weak morphisms and we show that the analogous problem for weak morphisms is even more complex than the original one. These considerations can be used among others to prove that the existence of codings is decidable for domain monoids whose dependence graphs are acyclic. On the other hand, we demonstrate that both problems are in general undecidable. In this extended abstract main ideas of these results are briefly sketched; we refer to the current version of the manuscript [8] for details.

In Section 2 we recall basic concepts and known results; see e.g. [4] for a more comprehensive introduction. We also define weak morphisms there and then we demonstrate some of their characteristic properties in Section 3. The aim of Section 4 is to describe a connection between the trace coding problem and its equivalent for weak morphisms. A proof of the undecidability result is outlined in Sections 5 to 7. First we show that it suffices to deal with the existence problem for weak morphisms with prescribed contents of images of letters and then we show how to encode the initial Post's correspondence problem (inPCP) into it.

* This research was partially supported by the Ministry of Education of the Czech Republic under the project MSM 143100009.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 603–614, 2001.

Let us fix notations. We mean by $\mathbb{N}$ the set of positive integers, by $|A|$ the cardinality of a set A and by $\operatorname{sym}\rho$ the symmetric closure of a binary relation ρ. The neutral element of any monoid is written as 1. We denote by Σ^* the monoid of words (free monoid) over a finite set Σ. In this context, Σ is often called an alphabet and its elements letters. For a word $w \in \Sigma^*$, we mean by $\operatorname{alph}(w)$ the set of all letters occurring in w (the content of w), by $|w|$ the length of w and by $|w|_x$ the number of occurrences of a letter $x \in \Sigma$ in w. The prefix ordering on Σ^* is denoted by $\preceq$. For $X \subseteq \Sigma$, let $\pi_X : \Sigma^* \to X^*$ be the projection morphism erasing all letters which are not in X (we write a list of letters rather than X).

2 Basic Concepts

Let Σ be a finite set and let I be an arbitrary symmetric and reflexive binary relation on Σ. We call I an *independence relation* on Σ and the undirected graph (Σ, I) an *independence alphabet*. The complement $D = (\Sigma \times \Sigma) \setminus I$ is called a *dependence relation* and the graph (Σ, D) a *dependence alphabet*. Usually independence relations are defined as irreflexive, but we adopt this notation since it faithfully corresponds to the behaviour of weak morphisms. In fact, the difference between strong and weak morphisms lies exactly in this modification.

Let $\sim_I$ be the congruence of the free monoid Σ^* generated by the relation $\{(xy, yx) \mid (x, y) \in I\}$. The quotient monoid $\Sigma^*/\sim_I$ is denoted by $\mathbb{M}(\Sigma, I)$ and called a *trace monoid*. Elements of this monoid are called *traces*. In what follows, we do not distinguish between letters and their one-element congruence classes.

It is well known that two words $w, w' \in \Sigma^*$ represent the same trace in $\mathbb{M}(\Sigma, I)$ if and only if $|w|_x = |w'|_x$ for every $x \in \Sigma$ and $\pi_{x,y}(w) = \pi_{x,y}(w')$ for every $(x, y) \in D$. This characterization in particular shows that trace monoids are cancellative and that all notions such as a content, a length or a projection can be used also for traces.

The *initial alphabet* and the *final alphabet* of a trace are defined as follows. Let $\operatorname{init}(1) = \operatorname{fin}(1) = \emptyset$ and for $s \in \mathbb{M}(\Sigma, I)$, $s \neq 1$, let $\operatorname{init}(s) = \{\operatorname{first}(w) \mid w \in s\}$ and $\operatorname{fin}(s) = \{\operatorname{last}(w) \mid w \in s\}$, where $\operatorname{first}(w)$ is the first letter of the word w and $\operatorname{last}(w)$ is the last one. We say that traces $s, t \in \mathbb{M}(\Sigma, I)$ are *independent* if they satisfy $\operatorname{alph}(s) \times \operatorname{alph}(t) \subseteq I \setminus \operatorname{id}_\Sigma$. Notice that in such a case $st = ts$.

Since trace monoids are defined by presentations, every morphism of trace monoids (briefly trace morphism) $\varphi : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma', I')$ is uniquely determined by any mapping $\varphi_0 : \Sigma \to (\Sigma')^*$, such that $\varphi_0(x) \in \varphi(x)$, which satisfies

$$\forall (x, y) \in I : \ \varphi_0(x)\varphi_0(y) \sim_{I'} \varphi_0(y)\varphi_0(x) \ . \tag{1}$$

Conversely, every mapping $\varphi_0 : \Sigma \to (\Sigma')^*$ satisfying (1) extends to a morphism.

Following the terminology from the theory of free monoids, we call a trace morphism a *coding* if it is injective. In connection with decision problems of trace codings, two particular classes of morphisms were already considered: strong morphisms, introduced in [3], and cp-morphisms, introduced in [6] as morphisms associated with clique-preserving morphisms of independence alphabets. For the purpose of dealing with the general case, we generalize the latter notion and

we call the arising morphisms weak. This approach also suggests an alternative definition of cp-morphisms. A morphism $\varphi : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma', I')$ is called

- *strong* if $\forall (x, y) \in I \setminus \mathrm{id}_\Sigma :\ \mathrm{alph}(\varphi(x)) \cap \mathrm{alph}(\varphi(y)) = \emptyset$.
- *weak* if $\forall x \in \Sigma :\ \mathrm{alph}(\varphi(x)) \times \mathrm{alph}(\varphi(x)) \subseteq I'$.
- a *cp-morphism* if it is weak and $\forall x \in \Sigma,\ a \in \Sigma' :\ |\varphi(x)|_a \leq 1$.

To obtain a description analogous to the above one for general morphisms also for strong and weak morphisms, it is enough to replace (1) with respectively

$$\forall (x, y) \in I \setminus \mathrm{id}_\Sigma :\ \mathrm{alph}(\varphi_0(x)) \times \mathrm{alph}(\varphi_0(y)) \subseteq I' \setminus \mathrm{id}_{\Sigma'}\ , \tag{2}$$

$$\forall (x, y) \in I :\ \mathrm{alph}(\varphi_0(x)) \times \mathrm{alph}(\varphi_0(y)) \subseteq I'\ . \tag{3}$$

The (*strong, weak*) *trace coding problem* asks to decide for given two independence alphabets (Σ, I) and (Σ', I') whether there exists a (strong, weak) coding $\varphi : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma', I')$. The *trace code problem* asks to decide for a given morphism $\varphi : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma', I')$ whether it is a coding.

The trace code problem is well known to be undecidable even for strong morphisms when both monoids are fixed and $\mathbb{M}(\Sigma, I)$ is free (see e.g. [4]). The undecidability result in the case of cp-morphisms was established in [5] using substantially more complex construction; it also immediately follows from Proposition 5 below.

If problems of existence of trace codings are considered, the situation is very different. In the first place, unlike for the trace code problem, it is not clear whether the dual problem (non-existence of codings) is recursively enumerable. The two classical cases are simple: all finitely generated free monoids can be embedded into the one with two generators and for free commutative monoids injectivity coincides with linear independence of images of letters. In [1] these characterizations were generalized to all instances of the trace coding problem where the domain monoid is a direct product of free monoids. The strong trace coding problem turned out to be NP-complete due to the following result.

Proposition 1 ([6]). *There exists a strong coding $\varphi : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma', I')$ if and only if there exists a mapping $H : \Sigma \to 2^{\Sigma'}$ satisfying for every $x, y \in \Sigma$:*

$$H(x) \times H(y) \subseteq I' \setminus \mathrm{id}_{\Sigma'} \iff (x, y) \in I \setminus \mathrm{id}_\Sigma\ ,$$

$$H(x) \times H(y) \subseteq I' \implies (x, y) \in I\ .$$

The reason for the relative simplicity of the strong trace coding problem is that a strong coding can be easily constructed as soon as reasonable contents of images of letters are chosen (this choice is provided by a mapping H). To see this notice that in the image of a letter dependent letters may occur, which allows us to encode all of the information needed for the deciphering provided we can ensure that images of independent letters commute. But by the defining condition for strong morphisms, these images are even independent. On the other hand, if we consider weak morphisms, the image of any letter consists entirely of independent letters. So we have less opportunities to encode some information into images of letters under weak morphisms than under general morphisms and that is why the problem of existence of weak codings becomes even more complex than the one for general codings (see Corollary 1).

3 Weak Codings

The motive for considering weak morphisms is that, compared with general morphisms, they possess many properties which simplify their manipulation. Ideas of the proof of the main result are based on facts presented in this section.

First, one can see that the image of a free commutative monoid under a weak morphism employs only independent letters.

Lemma 1. *For every weak morphism* $\varphi : \mathbb{M}(\Sigma, \Sigma \times \Sigma) \to \mathbb{M}(\Sigma', I')$*, the set* $A = \bigcup \{\operatorname{alph}(\varphi(x)) \mid x \in \Sigma\}$ *forms a clique in the graph* (Σ', I')*. If* φ *is a coding, then* $|A| \geq |\Sigma|$*.*

We are interested mainly in minimal counter-examples to the coding property for a trace morphism $\varphi : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma', I')$, i.e. in pairs (s, t) of traces $s, t \in \mathbb{M}(\Sigma, I)$ such that $s \neq t$ and $\varphi(s) = \varphi(t)$ having minimal $|s| + |t|$. Notice that each such counter-example satisfies $\operatorname{init}(s) \cap \operatorname{init}(t) = \operatorname{fin}(s) \cap \operatorname{fin}(t) = \emptyset$; otherwise we can obtain a smaller one by cancellation. For weak morphisms we have another straightforward observation.

Lemma 2. *Let* $\varphi : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma', I')$ *be a weak morphism such that* $\varphi(xy) \neq \varphi(yx)$ *for every* $(x, y) \in D$*. If* $s, t \in \mathbb{M}(\Sigma, I)$ *satisfy* $\varphi(s) = \varphi(t)$*, then* $\operatorname{init}(s) \times \operatorname{init}(t) \subseteq I$*.*

When trying to disprove the coding property, one constructs potential initial parts of counter-examples and then tries to extend them. For this, one has to know which pairs of traces can be prolonged by appending new letters to achieve their equality. As the next lemma shows, this can be done just when those parts of these traces which do not belong to their common prefix are independent.

Lemma 3. *Let* $s, t \in \mathbb{M}(\Sigma, I)$*. Then there exist traces* $u, v \in \mathbb{M}(\Sigma, I)$ *such that* $su = tv$ *if and only if for every* $(x, y) \in D$ *either* $\pi_{x,y}(s) \preceq \pi_{x,y}(t)$ *or* $\pi_{x,y}(t) \preceq \pi_{x,y}(s)$*. In such a case, the traces* s' *and* t' *resulting from* s *and* t *by taking just the first* $\min\{|s|_x, |t|_x\}$ *occurrences of each letter* $x \in \Sigma$ *are equal and there exist unique traces* $t\backslash s, s\backslash t \in \mathbb{M}(\Sigma, I)$ *such that* $s = s' \cdot (t\backslash s)$*,* $t = s' \cdot (s\backslash t)$ *and* $\operatorname{alph}(t\backslash s) \times \operatorname{alph}(s\backslash t) \subseteq I \setminus \operatorname{id}_\Sigma$*. In particular,* $s \cdot (s\backslash t) = t \cdot (t\backslash s)$*.*

The following notions are introduced in order to formalize reasoning about initial parts of counter-examples. For $s, t \in \mathbb{M}(\Sigma, I)$, we call (s, t) a *semi-equality* for a morphism $\varphi : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma', I')$ whenever $\operatorname{init}(s) \cap \operatorname{init}(t) = \emptyset$ and there exist traces $u, v \in \mathbb{M}(\Sigma', I')$ such that $\varphi(s)\, u = \varphi(t)\, v$. We say that it is *non-trivial* if there are no traces $s', t' \in \mathbb{M}(\Sigma, I)$ such that $ss' = tt'$. The pair $(\varphi(t)\backslash\varphi(s), \varphi(s)\backslash\varphi(t))$ is called the *state* of (s, t).

Lemma 4. *Let* $\varphi : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma', I')$ *be a morphism and let* $s, t \in \mathbb{M}(\Sigma, I)$ *be two traces satisfying* $s \neq t$ *and* $\varphi(s) = \varphi(t)$ *such that* $|s| + |t|$ *is minimal. If* $s = us'$ *and* $t = vt'$*, where* $u, v, s', t' \in \mathbb{M}(\Sigma, I)$*, then* (u, v) *is a semi-equality.*

All of the information one needs to explore possible continuations of a semi-equality is contained in its state. For weak morphisms, Lemma 4 can be partially reversed, namely, if we have a semi-equality whose state consists of independent letters, then it can be prolonged into a counter-example.

Lemma 5. *Let* $\varphi : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma', I')$ *be a weak morphism such that there exists a non-trivial semi-equality* (s,t) *for* φ *with a state* (u,v) *satisfying* $\mathrm{alph}(uv) \times \mathrm{alph}(uv) \subseteq I'$. *Then* φ *is not a coding.*

Proof. Denoting for any trace r by $r^{\triangleleft}$ the trace consisting of mirror images of elements of r, we have $\varphi(st^{\triangleleft}) = \varphi(s)(\varphi(t))^{\triangleleft} = \varphi(t)(\varphi(s))^{\triangleleft} = \varphi(ts^{\triangleleft})$ since $uv^{\triangleleft} = vu^{\triangleleft}$ due to the assumption. □

Let us now describe one method of extending a semi-equality for a weak morphism. It is based on modifying the semi-equality to make it suitable for appending a given pair of elements of Σ to its sides. We demonstrate how to use this construction to remove some letter from the state. This action usually results in a replacement of this letter by another one together with some effect on the rest of the state.

Let $n \in \mathbb{N}$. For $x \in \Sigma$, an *x-block* of length n in a trace $s \in \mathbb{M}(\Sigma, I)$ is a triple (t, x^n, t'), where $t, t' \in \mathbb{M}(\Sigma, I)$, such that $s = tx^n t'$, $\mathrm{fin}(t) \times \{x\} \subseteq D$ and $x \notin \mathrm{init}(t')$. In fact, an x-block is nothing but the maximal number of occurrences of x which can be grouped together in a given position in the trace. Notice that each occurrence of x in s lies in exactly one x-block. For every independence alphabet (Σ, I), let $\sigma_n : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma, I)$ denote the weak coding defined by $\sigma_n(x) = x^n$ for every $x \in \Sigma$.

Lemma 6. *Let* $\varphi : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma', I')$ *be a weak morphism and* (s,t) *a semi-equality for* φ *with a state* (u,v). *Then, for any* $n \in \mathbb{N}$, $(\sigma_n(s), \sigma_n(t))$ *is also a semi-equality for* φ *with the state* $(\sigma_n(u), \sigma_n(v))$.

Let in addition $x, y \in \Sigma$, $a \in \mathrm{init}(v) \cap \mathrm{alph}(\varphi(x)) \setminus \mathrm{alph}(\varphi(y))$ *be letters which satisfy* $(\mathrm{alph}(u \cdot \varphi(x)) \setminus \{a\}) \times \mathrm{alph}(v \cdot \varphi(y)) \subseteq I'$, $\mathrm{alph}(s) \times \{x\} \not\subseteq I$ *and* $\mathrm{alph}(t) \times \{y\} \not\subseteq I$. *Let* m *be the length of the first* a*-block in* v *and* $n = |\varphi(x)|_a$. *Then* $(\sigma_n(s) \cdot x^m, \sigma_n(t) \cdot y^m)$ *is a semi-equality for* φ *and its state* (u', v') *satisfies* $a \notin \mathrm{init}(v')$.

4 Reduction into the Weak Coding Equivalent

In this section we establish a connection between the trace coding problem and the weak trace coding problem. It is based on the standard decomposition of traces into primitive roots. Let us first recall basic facts about this construction. A trace $s \in \mathbb{M}(\Sigma, I) \setminus \{1\}$ is called *connected* if the subgraph of (Σ, D) induced by $\mathrm{alph}(s)$ is connected. It is called *primitive* if it is connected and $s = t^n$ with $t \in \mathbb{M}(\Sigma, I)$ implies $n = 1$. It is well known that every connected trace s is a power of a unique primitive trace called the *primitive root* of s. Clearly, every trace $s \in \mathbb{M}(\Sigma, I)$ can be uniquely decomposed into a product of independent connected traces called *connected components* of s.

Proposition 2 ([7]). *Let* $s, s' \in \mathbb{M}(\Sigma, I)$ *satisfy* $ss' = s's$ *and let* t *and* t' *be primitive roots of any connected components of* s *and* s', *respectively. Then either* t *and* t' *are independent or* $t = t'$.

For an arbitrary morphism $\varphi : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma', I')$, we consider for every letter $x \in \Sigma$ the decomposition of $\varphi(x)$ into primitive roots of its connected components. By Proposition 2 primitive traces do not commute unless they are equal or independent and therefore the substantial information characterizing their behaviour is their content. So, we introduce sufficiently many new letters for each possible content and replace these primitive roots with them. As we never use more than one primitive root with a given content in one image, for fixed alphabets (Σ, I) and (Σ', I') we can manage with a finite number of new letters. In this way we express every morphism φ as a composition of a weak morphism and a strong morphism. Clearly, if φ is a coding, the arising weak morphism is a coding as well. On the other hand, we can use Proposition 1 to find a strong coding for prolonging every weak coding to the new codomain monoid into a coding to the original one.

More precisely, we denote by $\mathcal{C}(\Sigma', D')$ the set of all subsets $A \subseteq \Sigma'$, $|A| \geq 2$, such that the subgraph of (Σ', D') induced by A is connected. We define a new independence alphabet $(\Sigma'_\Sigma, I'_\Sigma)$ as follows. Let $\Sigma'_\Sigma = \Sigma' \cup (\mathcal{C}(\Sigma', D') \times \Sigma)$ and for $a, b \in \Sigma'$, $A, B \in \mathcal{C}(\Sigma', D')$, $x, y \in \Sigma$:

$$\begin{aligned} a\ I'_\Sigma\ b &\iff a\ I'\ b\ , \\ a\ I'_\Sigma\ (A, x) &\iff \{a\} \times A \subseteq I'\ , \\ (A, x)\ I'_\Sigma\ (B, y) &\iff A \times B \subseteq I' \text{ or } (A, x) = (B, y)\ . \end{aligned}$$

Proposition 3. *Let (Σ, I) and (Σ', I') be independence alphabets. Then there exists a coding from $\mathbb{M}(\Sigma, I)$ to $\mathbb{M}(\Sigma', I')$ if and only if there exists a weak coding from $\mathbb{M}(\Sigma, I)$ to $\mathbb{M}(\Sigma'_\Sigma, I'_\Sigma)$.*

As an immediate consequence we obtain

Corollary 1. *There exists an effective reduction of the trace coding problem into the weak trace coding problem.*

Notice that in Proposition 3 there is the same domain monoid on both sides of the equivalence. This makes it suitable for showing the decidability of the trace coding problem for some classes of instances specified by properties of the domain monoid. For example, the following result can be obtained by means of proving its weak coding equivalent.

Theorem 1. *The trace coding problem restricted to instances whose domain monoids are defined by acyclic dependence alphabets is decidable.*

5 Content Fixation

The aim of this section is to describe how the problem of existence of weak codings satisfying certain requirements on contents of images of letters can be reduced to the trace coding problem.

We use two mappings to specify restrictions on contents – one of them to express which letters are compulsory and the other to express which are allowed.

Let $\mu, \nu : \Sigma \to 2^{\Sigma'}$ be mappings. A weak morphism $\varphi : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Sigma', I')$ is called (μ, ν)-*weak* if it satisfies $\mu(x) \subseteq \mathrm{alph}(\varphi(x)) \subseteq \nu(x)$ for every $x \in \Sigma$. It is called ν-*weak* whenever $\mathrm{alph}(\varphi(x)) \subseteq \nu(x)$ for all $x \in \Sigma$.

Proposition 4. *Let (Σ, I), (Σ', I') be independence alphabets such that $D \neq \emptyset$ and I is transitive. Let $\mu, \nu : \Sigma \to 2^{\Sigma'}$ be mappings satisfying for all $x, y \in \Sigma$:*

$$x \; I \; y, \;\; x \neq y \implies \mu(x) = \mu(y) = \emptyset \;,$$
$$\big(\nu(x) \times \nu(x) \subseteq I'\big) \;\&\; \big(x \; I \; y \implies \nu(x) = \nu(y)\big) \;.$$

Then one can effectively construct independence alphabets (Σ_1, I_1) and (Σ_1', I_1') such that the following statements are equivalent.

1. *There exists a (μ, ν)-weak coding from $\mathbb{M}(\Sigma, I)$ to $\mathbb{M}(\Sigma', I')$.*
2. *There exists a weak coding from $\mathbb{M}(\Sigma_1, I_1)$ to $\mathbb{M}(\Sigma_1', I_1')$.*
3. *There exists a weak coding from $\mathbb{M}(\Sigma_1, I_1)$ to $\mathbb{M}\big((\Sigma_1')_{\Sigma_1}, (I_1')_{\Sigma_1}\big)$.*
4. *There exists a coding from $\mathbb{M}(\Sigma_1, I_1)$ to $\mathbb{M}(\Sigma_1', I_1')$.*

The proof of this result consists of two major steps.

First, we specify mandatory letters defined by μ using only ν. There is nothing to take care of for letters $x \in \Sigma$ such that $|\nu(x)| = 1$ since $\mathrm{alph}(\varphi(x)) = \nu(x)$ is satisfied for every ν-weak coding φ. The idea of the construction is to enrich each of the original alphabets with a set Θ of new letters and define $\nu(y) = \{y\}$ for every $y \in \Theta$; as the behaviour of any ν-weak coding on these letters is obvious, it can serve as a skeleton for prescribing contents of images of other letters. More precisely, to ensure that the image of x under every ν-weak coding contains a, we introduce a letter $(x, a) \in \Theta$ dependent on x in the domain alphabet and dependent only on a in the codomain alphabet.

Second, we show how to manage our content requirements even without a mapping ν. This time, we add to the alphabets mutually dependent cliques of independent letters, each of them having sufficiently distinct size. Then we can employ Lemma 1 to verify that images of letters of a given clique under a weak coding use almost exclusively letters from the clique of the same size. So, in order to deal with the requirements for a letter $x \in \Sigma$, we introduce a clique having all elements independent on x in the domain alphabet and independent exactly on letters allowed in the image of x in the codomain alphabet. As images of independent letters under a weak morphism contain only independent ones, this ensures that prohibited letters are never used.

In effect the latter construction functions in the same way even if we add new letters according to the previous section in order to obtain the third condition. It is due to the fact that these new letters corresponding to connected subgraphs of the codomain dependence alphabet do not form bigger cliques of independence than those already existing.

Finally, we use Proposition 3 to deduce the equivalence of conditions 3 and 4.

6 Encoding of the PCP

It is well known that the dual problem to the inPCP (co-inPCP) is not recursively enumerable. In this section we construct a reduction of the co-inPCP into the problem of existence of (μ, ν)-weak codings. As our construction should be based only on contents of images of letters, we have to impose certain restriction on instances of the inPCP, which enables us not to care about powers of letters in these images.

Let us denote by $\mathcal{P}$ the following instance of the inPCP. Let $n \in \mathbb{N}$. We are given n pairs $(w_1, \overline{w}_1), \ldots, (w_n, \overline{w}_n)$ of non-empty words over a finite alphabet Ξ such that every product of $w_1, \overline{w}_1, \ldots, w_n, \overline{w}_n$ contains no subword of the form x^k with $x \in \Xi$ and $k \geq 2$. The problem asks to decide whether there exists some finite sequence $i_1, \ldots, i_m$ of natural numbers from the set $\{1, \ldots, n\}$ satisfying $w_1 w_{i_1} w_{i_2} \cdots w_{i_m} = \overline{w}_1 \overline{w}_{i_1} \overline{w}_{i_2} \cdots \overline{w}_{i_m}$.

Notice that this restriction on instances causes no loss of generality since for every instance of the inPCP we can obtain an equivalent instance of the above form by introducing a new letter # into Ξ and performing the substitution $x \mapsto x\#$ for all $x \in \Xi$.

For $i = 1, \ldots, n$ and $j = 1, \ldots, |w_i|$ and $k = 1, \ldots, |\overline{w}_i|$, we refer to the j-th letter of the word w_i as x_{ij} and to the k-th letter of $\overline{w}_i$ as $\overline{x}_{ik}$. For the rest of this section, when writing indices i, ij or ik, we implicitly assume that they run through all values as in the previous sentence.

We define two independence alphabets (Σ, I) and (Σ', I'). First, we introduce a set of new letters:

$$\begin{aligned}\Omega = \{&\alpha, \alpha_0, \beta_1, \ldots, \beta_8, \gamma_1, \gamma_2, \gamma_3, \delta_{i1}, \delta_{i2}, \varepsilon_{ij1}, \varepsilon_{ij2}, \overline{\varepsilon}_{ik1}, \overline{\varepsilon}_{ik2},\\ &\zeta_{ij1}, \zeta_{ij2}, \overline{\zeta}_{ik1}, \overline{\zeta}_{ik2}, \eta_{ij1}, \eta_{ij2}, \eta_{ij3}, \overline{\eta}_{ik1}, \overline{\eta}_{ik2}, \overline{\eta}_{ik3}, \vartheta_{ij}, \overline{\vartheta}_{ik},\\ &\iota_i, \kappa_1, \kappa_2, \lambda_{ij1}, \lambda_{ij2}, \lambda_{ij3}, \overline{\lambda}_{ik1}, \overline{\lambda}_{ik2}, \overline{\lambda}_{ik3}, \xi_{i1}, \xi_{i2}, \xi_{i3}\} \ .\end{aligned}$$

The domain alphabet $\Sigma = (\Omega \setminus \{\alpha_0\}) \times \{1, 2\}$ consists of one pair of letters for each element of $\Omega \setminus \{\alpha_0\}$. Letters from these pairs should appear on opposite sides of a counter-example to the coding property and correspond there to each other according to their first coordinates. Let $I = \mathrm{id}_\Sigma \cup \mathrm{sym}\,\{((\alpha, 1), (\alpha, 2))\}$.

In the outcome, counter-examples to the coding property should correspond to solutions of $\mathcal{P}$. A computation of a solution of $\mathcal{P}$ is simulated by adding new pairs from Σ to an already constructed semi-equality in the way determined by its state. Just one pair of letters in Σ is set independent to ensure that there is only one way to start this computation by Lemma 2.

The alphabet Σ' is divided into several disjoint subsets according to the role of letters in the encoding:

$$\Sigma' = \{l_1, l_2, r_1, r_2, b, c, d\} \cup S \cup A \cup E \cup F \cup G \cup P \cup Q \cup \Xi \cup \{\star\} \cup \Omega \ .$$

Elements of the set $S = \left\{s, s_f, s_i, s_{ij}, \overline{s}_{ik}, t_{ij}, \overline{t}_{ik}\right\}$ control a computation during its initial and final phase and keep it from a premature termination. The process of composing the words w_i and $\overline{w}_i$ is controlled by elements of $A = \{a, a_{ij}, \overline{a}_{ik}\}$;

a semi-equality with a_{ij} ($\overline{a}_{ik}$) in the state is extended by pairs introducing x_{ij} ($\overline{x}_{ik}$ respectively) and a in the state allows to choose the next pair $(w_i, \overline{w}_i)$. The progress of a computation is determined by dependences between a's and letters from the sets $E = \{e, e_{ij}, \overline{e}_{ik}\}$, $F = \{f, f_{ij}, \overline{f}_{ik}\}$, $G = \{g_i, g_{ij}, \overline{g}_{ik}\}$, $P = \{p_{ij}, \overline{p}_{ik}\}$ and $Q = \{q_i, q_{ij}, \overline{q}_{ik}\}$. The letter $\star$ behaves in the same way as letters of Ξ; it should mark the end of a solution of $\mathcal{P}$. Letters from Σ are placed on the appropriate sides of a semi-equality thanks to l's and r's and the pairs of letters in Σ are fixed using elements of Ω.

In the following we set dependences between elements of Σ', all pairs not mentioned below are considered independent:

$$l_1 \; D' \; l_2, \;\; r_1 \; D' \; r_2, \;\; b \; D' \; c, \;\; b \; D' \; d \;,$$
$$p_{ij} \; D' \; e_{ij}, \;\; \overline{p}_{ik} \; D' \; \overline{e}_{ik}, \;\; q_i \; D' \; f, \;\; q_{ij} \; D' \; f_{ij}, \;\; \overline{q}_{ik} \; D' \; \overline{f}_{ik} \;,$$

$$I' \cap S^2 = \mathrm{id}_S, \quad I' \cap (\Xi \cup \{\star\})^2 = \mathrm{id}_{\Xi \cup \{\star\}} \;,$$
$$I' \cap \Omega^2 = \mathrm{id}_\Omega \cup \mathrm{sym}\,(\{\alpha, \alpha_0\} \times \{\beta_1, \beta_2, \beta_3, \beta_4\}) \;,$$

$$\begin{aligned} I' \cap (A \cup E \cup F \cup G)^2 = \mathrm{id}_{A \cup E \cup F \cup G} \cup \mathrm{sym}\{&(e, a), (e_{ij}, a_{ij}), (\overline{e}_{ik}, \overline{a}_{ik}), \\ &(f, a), (f_{ij}, a_{ij}), (\overline{f}_{ik}, \overline{a}_{ik}), (g_i, a), (g_i, a_{i1}), (g_{ij}, a_{ij}), (g_{ij}, a_{ij+1}), \\ &(\overline{g}_{ik}, \overline{a}_{ik}), (\overline{g}_{ik}, \overline{a}_{ik+1}), (g_i, f_{i1}), (g_{ij}, f_{ij+1}), (\overline{g}_{ik}, \overline{f}_{ik+1})\} \;, \end{aligned}$$

where $a_{i|w_i|+1} = \overline{a}_{i1}$, $\overline{a}_{i|\overline{w}_i|+1} = a$, $f_{i|w_i|+1} = \overline{f}_{i1}$ and $\overline{f}_{i|\overline{w}_i|+1} = f$.

Now we are going to construct mappings $\mu, \nu : \Sigma \to 2^{\Sigma'}$ which satisfy the assumptions of Proposition 4. For each $\omega \in \Omega \setminus \{\alpha_0\}$, the images of $(\omega, 1)$ and $(\omega, 2)$ under μ and ν are understood as one rule for a computation. As the desired contents are given by ν, let us define ν first.

The following rules guarantee that a computation starts correctly and that we can remove control letters at the end of a successful computation:

$$\begin{aligned} \nu((\alpha,1)) &= \{l_1, r_1, b, s_1, a_{11}, \alpha\} & \nu((\alpha,2)) &= \{l_1, r_1, b, s_1, a_{11}, \alpha\} \\ \nu((\beta_1,1)) &= \{l_2, r_1, \alpha_0, \beta_1\} & \nu((\beta_1,2)) &= \{l_2, r_2, \beta_1\} \\ \nu((\beta_2,1)) &= \{l_2, \alpha_0, \beta_2\} & \nu((\beta_2,2)) &= \{l_2, r_2, \alpha, \beta_2\} \\ \nu((\beta_3,1)) &= \{l_1, r_2, \alpha_0, \beta_3\} & \nu((\beta_3,2)) &= \{l_2, r_2, \beta_3\} \\ \nu((\beta_4,1)) &= \{r_2, \alpha_0, \beta_4\} & \nu((\beta_4,2)) &= \{l_2, r_2, \alpha, \beta_4\} \\ \nu((\beta_5,1)) &= \{l_2, s_f, \beta_5\} & \nu((\beta_5,2)) &= \{r_2, s_1, \beta_5\} \\ \nu((\beta_6,1)) &= \{l_2, s_1, \beta_6\} & \nu((\beta_6,2)) &= \{r_2, s_f, a, \beta_6\} \\ \nu((\beta_7,1)) &= \{l_2, c, \beta_7\} & \nu((\beta_7,2)) &= \{r_2, b, \beta_7\} \\ \nu((\beta_8,1)) &= \{l_2, b, \beta_8\} & \nu((\beta_8,2)) &= \{r_2, d, \beta_8\} \\ \nu((\gamma_1,1)) &= \{l_2, s_f, \gamma_1\} & \nu((\gamma_1,2)) &= \{l_1, s_f, \gamma_1\} \\ \nu((\gamma_2,1)) &= \{r_1, s_f, \gamma_2\} & \nu((\gamma_2,2)) &= \{r_2, s_f, \gamma_2\} \\ \nu((\gamma_3,1)) &= \{a_{11}, s_f, \gamma_3\} & \nu((\gamma_3,2)) &= \{a, s_f, \gamma_3\} \;. \end{aligned}$$

The next family of rules serves for setting initial powers of p's and q's:

$$\begin{aligned}
\nu((\delta_{i1},1)) &= \{l_2, c, s_i, \delta_{i1}\} & \nu((\delta_{i1},2)) &= \{r_2, d, s_{i+1}, q_i, \delta_{i1}\}\\
\nu((\delta_{i2},1)) &= \{l_2, c, s_i, q_i, \delta_{i2}\} & \nu((\delta_{i2},2)) &= \{r_2, d, s_{i+1}, \delta_{i2}\}\\
\nu((\varepsilon_{ij1},1)) &= \{l_2, s_{ij}, \varepsilon_{ij1}\} & \nu((\varepsilon_{ij1},2)) &= \{r_2, s_{ij+1}, p_{ij}, \varepsilon_{ij1}\}\\
\nu((\varepsilon_{ij2},1)) &= \{l_2, s_{ij}, p_{ij}, \varepsilon_{ij2}\} & \nu((\varepsilon_{ij2},2)) &= \{r_2, s_{ij+1}, \varepsilon_{ij2}\}\\
\nu((\overline{\varepsilon}_{ik1},1)) &= \{l_2, \overline{s}_{ik}, \overline{\varepsilon}_{ik1}\} & \nu((\overline{\varepsilon}_{ik1},2)) &= \{r_2, \overline{s}_{ik+1}, \overline{p}_{ik}, \overline{\varepsilon}_{ik1}\}\\
\nu((\overline{\varepsilon}_{ik2},1)) &= \{l_2, \overline{s}_{ik}, \overline{p}_{ik}, \overline{\varepsilon}_{ik2}\} & \nu((\overline{\varepsilon}_{ik2},2)) &= \{r_2, \overline{s}_{ik+1}, \overline{\varepsilon}_{ik2}\}\\
\nu((\zeta_{ij1},1)) &= \{l_2, t_{ij}, \zeta_{ij1}\} & \nu((\zeta_{ij1},2)) &= \{r_2, t_{ij+1}, q_{ij}, \zeta_{ij1}\}\\
\nu((\zeta_{ij2},1)) &= \{l_2, t_{ij}, q_{ij}, \zeta_{ij2}\} & \nu((\zeta_{ij2},2)) &= \{r_2, t_{ij+1}, \zeta_{ij2}\}\\
\nu((\overline{\zeta}_{ik1},1)) &= \{l_2, \overline{t}_{ik}, \overline{\zeta}_{ik1}\} & \nu((\overline{\zeta}_{ik1},2)) &= \{r_2, \overline{t}_{ik+1}, \overline{q}_{ik}, \overline{\zeta}_{ik1}\}\\
\nu((\overline{\zeta}_{ik2},1)) &= \{l_2, \overline{t}_{ik}, \overline{q}_{ik}, \overline{\zeta}_{ik2}\} & \nu((\overline{\zeta}_{ik2},2)) &= \{r_2, \overline{t}_{ik+1}, \overline{\zeta}_{ik2}\}\ ,
\end{aligned}$$

where

$$s_{n+1} = s_{11},\ s_{i|w_i|+1} = s_{i+11},\ s_{n+11} = \overline{s}_{11},\ \overline{s}_{i|\overline{w}_i|+1} = \overline{s}_{i+11},\ \overline{s}_{n+11} = t_{11},$$
$$t_{i|w_i|+1} = t_{i+11},\ t_{n+11} = \overline{t}_{11},\ \overline{t}_{i|\overline{w}_i|+1} = \overline{t}_{i+11},\ \overline{t}_{n+11} = s\ .$$

The main cycle inserting letters from Ξ is performed by:

$$\begin{aligned}
\nu((\eta_{ij1},1)) &= \{l_2, s, p_{ij}, x_{ij}, \eta_{ij1}\} & \nu((\eta_{ij1},2)) &= \{r_2, s, e_{ij}, \eta_{ij1}\}\\
\nu((\eta_{ij2},1)) &= \{l_2, s, e_{ij}, \eta_{ij2}\} & \nu((\eta_{ij2},2)) &= \{r_2, s, p_{ij}, \eta_{ij2}\}\\
\nu((\eta_{ij3},1)) &= \{l_2, s, p_{ij}, \eta_{ij3}\} & \nu((\eta_{ij3},2)) &= \{r_2, s, e_{ij}, \eta_{ij3}\}\\
\nu((\overline{\eta}_{ik1},1)) &= \{l_2, s, \overline{p}_{ik}, \overline{\eta}_{ik1}\} & \nu((\overline{\eta}_{ik1},2)) &= \{r_2, s, \overline{e}_{ik}, \overline{x}_{ik}, \overline{\eta}_{ik1}\}\\
\nu((\overline{\eta}_{ik2},1)) &= \{l_2, s, \overline{e}_{ik}, \overline{\eta}_{ik2}\} & \nu((\overline{\eta}_{ik2},2)) &= \{r_2, s, \overline{p}_{ik}, \overline{\eta}_{ik2}\}\\
\nu((\overline{\eta}_{ik3},1)) &= \{l_2, s, \overline{p}_{ik}, \overline{\eta}_{ik3}\} & \nu((\overline{\eta}_{ik3},2)) &= \{r_2, s, \overline{e}_{ik}, \overline{\eta}_{ik3}\}\\
\nu((\vartheta_{ij},1)) &= \{l_2, s, a_{ij+1}, x_{ij}, \vartheta_{ij}\} & \nu((\vartheta_{ij},2)) &= \{r_2, s, a_{ij}, g_{ij}, \vartheta_{ij}\}\\
\nu((\overline{\vartheta}_{ik},1)) &= \{l_2, s, \overline{a}_{ik+1}, \overline{\vartheta}_{ik}\} & \nu((\overline{\vartheta}_{ik},2)) &= \{r_2, s, \overline{a}_{ik}, \overline{g}_{ik}, \overline{x}_{ik}, \overline{\vartheta}_{ik}\}\\
\nu((\iota_i,1)) &= \{l_2, s, a_{i1}, \iota_i\} & \nu((\iota_i,2)) &= \{r_2, s, a, g_i, \iota_i\}\ ,
\end{aligned}$$

where $a_{i|w_i|+1} = \overline{a}_{i1}$ and $\overline{a}_{i|\overline{w}_i|+1} = a$. A computation of a solution of $\mathcal{P}$ is successful if we can eventually use one of the following rules:

$$\begin{aligned}
\nu((\kappa_1,1)) &= \{l_2, s, \star, \kappa_1\} & \nu((\kappa_1,2)) &= \{r_2, s_f, e, \star, \kappa_1\}\\
\nu((\kappa_2,1)) &= \{l_2, s_f, \star, \kappa_2\} & \nu((\kappa_2,2)) &= \{r_2, s, e, \star, \kappa_2\}\ .
\end{aligned}$$

Finally, for manipulating letters from G we need the rules:

$$\begin{aligned}
\nu((\lambda_{ij1},1)) &= \{l_2, s, g_{ij-1}, q_{ij}, \lambda_{ij1}\} & \nu((\lambda_{ij1},2)) &= \{r_2, s, f_{ij}, \lambda_{ij1}\}\\
\nu((\lambda_{ij2},1)) &= \{l_2, s, f_{ij}, \lambda_{ij2}\}, & \nu((\lambda_{ij2},2)) &= \{r_2, s, q_{ij}, \lambda_{ij2}\}\\
\nu((\lambda_{ij3},1)) &= \{l_2, s, q_{ij}, \lambda_{ij3}\} & \nu((\lambda_{ij3},2)) &= \{r_2, s, f_{ij}, \lambda_{ij3}\}
\end{aligned}$$

$$\begin{aligned}
\nu((\overline{\lambda}_{ik1},1)) &= \{l_2, s, \overline{g}_{ik-1}, \overline{q}_{ik}, \overline{\lambda}_{ik1}\} & \nu((\overline{\lambda}_{ik1},2)) &= \{r_2, s, \overline{f}_{ik}, \overline{\lambda}_{ik1}\}\\
\nu((\overline{\lambda}_{ik2},1)) &= \{l_2, s, \overline{f}_{ik}, \overline{\lambda}_{ik2}\} & \nu((\overline{\lambda}_{ik2},2)) &= \{r_2, s, \overline{q}_{ik}, \overline{\lambda}_{ik2}\}\\
\nu((\overline{\lambda}_{ik3},1)) &= \{l_2, s, \overline{q}_{ik}, \overline{\lambda}_{ik3}\} & \nu((\overline{\lambda}_{ik3},2)) &= \{r_2, s, \overline{f}_{ik}, \overline{\lambda}_{ik3}\}\\
\nu((\xi_{i1},1)) &= \{l_2, s, \overline{g}_{i|\overline{w}_i|}, q_i, \xi_{i1}\} & \nu((\xi_{i1},2)) &= \{r_2, s, f, \xi_{i1}\}\\
\nu((\xi_{i2},1)) &= \{l_2, s, f, \xi_{i2}\} & \nu((\xi_{i2},2)) &= \{r_2, s, q_i, \xi_{i2}\}\\
\nu((\xi_{i3},1)) &= \{l_2, s, q_i, \xi_{i3}\} & \nu((\xi_{i3},2)) &= \{r_2, s, f, \xi_{i3}\}\ ,
\end{aligned}$$

where $g_{i0} = g_i$ and $\overline{g}_{i0} = g_{i|w_i|}$. The mapping $\mu : \Sigma \to 2^{\Sigma'}$ is defined by the same rules as ν except $\mu((\alpha,1)) = \mu((\alpha,2)) = \emptyset$.

Let φ be the (μ,ν)-weak morphism from $\mathbb{M}(\Sigma, I)$ to $\mathbb{M}(\Sigma', I')$ satisfying $\varphi((\alpha,1)) = l_1 b a_{11} \alpha$, $\varphi((\alpha,2)) = r_1 b s_1 \alpha$ and for all $\omega \in \Omega \setminus \{\alpha, \alpha_0\}$, $z \in \Sigma'$ and $h \in \{1,2\}$:

$$|\varphi((\omega,h))|_z = \begin{cases} 1 & \text{if } z \in \nu((\omega,h))\ , \\ 0 & \text{otherwise .} \end{cases}$$

Proposition 5. *Let $\mathcal{P}$, (Σ, I), (Σ', I'), μ, ν and φ be as defined above. Then the following statements are equivalent.*

1. *$\mathcal{P}$ has no solution.*
2. *φ is a coding.*
3. *There exists a (μ,ν)-weak coding from $\mathbb{M}(\Sigma, I)$ to $\mathbb{M}(\Sigma', I')$.*

To prove this result, one has to find a solution of $\mathcal{P}$ using a minimal counter-example (u,v), $u, v \in \mathbb{M}(\Sigma, I)$, to the coding property for φ and conversely to construct from a solution of $\mathcal{P}$ such a counter-example for every (μ,ν)-weak morphism from $\mathbb{M}(\Sigma, I)$ to $\mathbb{M}(\Sigma', I')$.

Since elements of $\Omega \setminus \{\alpha, \alpha_0\}$ are pairwise dependent and in the φ-image of every letter except $(\alpha,1)$ and $(\alpha,2)$ there is just one occurrence of exactly one of them, letters in u and v must be paired according to them. When we go through semi-equalities respecting these pairs obtained from u and v using Lemma 4, it can be proved that (except for several initial ones) one side of the state always starts on r_1 and contains s and the other starts on l_1 and contains some letter from A which changes when pairs of ϑ's, $\overline{\vartheta}$'s or ι's are added. This happens in such an order that the accumulation of letters from Ξ in the images of the sides of these semi-equalities simulates a multiplication of pairs of words from the instance $\mathcal{P}$ – the result differs just in powers of letters, which does not matter due to our assumption on the instance. The letter s vanishes from the state only when a pair of κ's is used. But κ's can appear just when a is in the state thanks to the occurrences of e in φ-images of κ's, that is right after completing an addition of some pair of words from $\mathcal{P}$. As all images of κ's contain $\star$, we deduce that the projections of the images of the sides of the semi-equality to Ξ are equal and therefore correspond to a solution of $\mathcal{P}$.

As for the converse, it can be shown that it is sufficient to consider only morphisms allowing the same arguments about the pairs of letters from Σ as above. We start with a suitable number of α's on both sides and inductively

construct semi-equalities similar to those for φ by appending new pairs using Lemma 6. During the main cycle of a computation, we just have to take care of letters from Ξ to ensure that the corresponding blocks of these letters in the images of both sides of these semi-equalities have the same length. This is done by iterative addition of pairs of η's or $\overline{\eta}$'s replacing p's in the state with e's and vice versa, which modifies the number of occurrences of the corresponding letter from Ξ in the state. If $\mathcal{P}$ has a solution, some pair of κ's can be eventually used to replace all occurrences of s in the state with s_f. Then one can easily remove all dependent letters from the state using γ's and apply Lemma 5.

7 Conclusion

Main Theorem. *The trace coding problem is not recursively enumerable.*

Proof. We construct an effective reduction of the co-inPCP into our problem. First, we utilize Proposition 5 to reduce the co-inPCP into deciding the existence of (μ, ν)-weak codings. Since all instances (consisting of (Σ, I), (Σ', I'), μ and ν) constructed there satisfy assumptions of Proposition 4, we can use it to prolong the reduction into the trace coding problem. □

The same result for weak morphisms immediately follows due to Corollary 1.

Corollary 2. *The weak trace coding problem is not recursively enumerable.*

Acknowledgement. I am grateful to Jiří Srba and one of the referees for their useful suggestions.

References

1. Bruyère, V., De Felice, C.: Coding and strong coding in trace monoids. In *Proc. STACS'95*, LNCS 900, Springer (1995) 373–384.
2. Bruyère, V., De Felice, C.: On the existence of codings between trace monoids. *Journal of Automata, Languages and Combinatorics* **4** (1999) 87–100.
3. Bruyère, V., De Felice, C., Guaiana, G.: On some decision problems for trace codings. *Theoretical Computer Science* **148** (1995) 227–260.
4. Diekert, V., Métivier, Y.: Partial commutation and traces. In *Handbook of Formal Languages, Vol. 3*, Springer (1997) 457–533.
5. Diekert, V., Muscholl, A.: Code problems on traces. In *Proc. MFCS'96*, LNCS 1113, Springer (1996) 2–17.
6. Diekert, V., Muscholl, A., Reinhardt, K.: On codings of traces. In *Proc. STACS'95*, LNCS 900, Springer (1995) 385–396.
7. Duboc, C.: On some equations in free partially commutative monoids. *Theoretical Computer Science* **46** (1986) 159–174.
8. Kunc, M.: Undecidability of the trace coding problem and some decidable cases. manuscript (2001). `http://www.math.muni.cz/~kunc`
9. Mazurkiewicz, A.: Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus (1977).
10. Ochmański, E.: On morphisms of trace monoids. In *Proc. STACS'88*, LNCS 294, Springer (1988) 346–355.

Combinatorics of Periods in Strings

Eric Rivals[1] and Sven Rahmann[2]

[1] L.I.R.M.M., CNRS U.M.R. 5506
161 rue Ada, F-34392 Montpellier Cedex 5, France,
rivals@lirmm.fr

[2] Max-Planck-Institut für Molekulare Genetik,
Dept. of Computational Molecular Biology,
Ihnestraße 73, D-14195 Berlin, Germany
rahmann@molgen.mpg.de

Abstract. We consider the set $\Gamma(n)$ of all period sets of strings of length n over a finite alphabet. We show that there is redundancy in period sets and introduce the notion of an irreducible period set. We prove that $\Gamma(n)$ is a lattice under set inclusion and does not satisfy the Jordan-Dedekind condition. We propose the first enumeration algorithm for $\Gamma(n)$ and improve upon the previously known asymptotic lower bounds on the cardinality of $\Gamma(n)$. Finally, we provide a new recurrence to compute the number of strings sharing a given period set.

1 Introduction

We consider the period sets of strings of length n over a finite alphabet, and specific representations of them, *(auto)correlations*, which are binary vectors of length n indicating the periods. Among the possible 2^n bit vectors, only a small subset are valid autocorrelations. In [6], Guibas and Odlyzko provide characterizations of correlations, asymptotic bounds on their number, and a recurrence for the *population size* of a correlation, i.e., the number of strings sharing a given correlation. However, until now, no one has investigated the combinatorial structure of $\Gamma(n)$, the set of all correlations of length n; nor has anyone proposed an efficient enumeration algorithm for $\Gamma(n)$.

In this paper, we show that there is redundancy in period sets, introduce the notion of an *irreducible period set*, and show how to efficiently convert between the two representations (Section 2). We prove that $\Gamma(n)$ is a lattice under set inclusion and does not satisfy the Jordan-Dedekind condition. While $\Lambda(n)$, the set of all irreducible period sets, does satisfy that condition, it does not form a lattice (Section 3). We propose the first enumeration algorithm for $\Gamma(n)$ (Section 4) and improve upon the previously known asymptotic lower bounds for the cardinality of $\Gamma(n)$ (Section 5). Finally, we provide a new recurrence to compute the population sizes of correlations (Section 6).

Periods of strings have proven useful mainly in two areas of research. First, in pattern matching, several off-line algorithms take advantage of the periods of the pattern to speed up the search for its occurrences in a text (see [2] for a review). Second, several statistics of pattern occurrences have been investigated

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 615–626, 2001.

which take into account the pattern's periodicity. For instance, the probability of a pattern's absence in a Bernoulli text depends on its correlation [9]. In another work [8], we investigate the number of missing words in a random text and the number of common words between two random texts. Computing their expectation requires the enumeration of all correlations and the calculation of their population sizes. This has applications in the analysis of approximate pattern matching, in computational molecular biology, and in the testing of random number generators.

1.1 Notations, Definitions, and Elementary Properties

Let Σ be a finite alphabet of size σ. A sequence of n letters of Σ indexed from 0 to $n-1$ is called a *word* or a *string* of length n over Σ. We denote the *length* of a word $U := U_0 U_1 \dots U_{n-1}$ by $|U|$. For any $0 \le i \le j < n$, $U_{i..j} := U_i \dots U_j$ is called a *substring* of U. Moreover, $U_{0..j}$ is a *prefix* and $U_{i..n-1}$ is a *suffix* of U. We denote by Σ^*, respectively by Σ^n, the set of all finite words, resp. of all words of length n, over Σ.

Definition 1 (Period). Let $U \in \Sigma^n$ and let p be a non-negative integer with $p < n$. Then p is a *period* of U iff: $\forall\, 0 \le i < n - p \ : U_i = U_{i+p}$.

In other words, p is a period iff another copy of U shifted p positions to the right over the original matches in the overlapping positions, or equivalently, iff the prefix and suffix of U of length $n-p$ are equal. By convention, any word has the trivial null period, 0.

Some properties of periods are: If p is a period then any multiple of p lower than n is also period. If p is a period and the suffix of length $n-p$ has period q, then U has period $p+q$, and conversely. For an in-depth study, we refer the reader to [1,7,6]. Here, we need the Theorem of Fine and Wilf, also called the GCD-rule, and a useful corollary.

Theorem 1 (Fine and Wilf [4]). *Let $U \in \Sigma^n$. If U has periods p and q with $p \le q$ and $p + q \le n + \gcd(p,q)$, then $\gcd(p,q)$ is also a period.*

Lemma 1. *Let $U \in \Sigma^n$ with smallest non-null period $p \le \lfloor \frac{n}{2} \rfloor$. If $i < n - p + 2$ is a period of U, then it is a multiple of p.*

Proof. Assume that $p \nmid i$. Then $g := \gcd(p,i) < p$, and trivially $g \ge 1$. Therefore, $p + i - g \le n$, and Theorem 1 says that g is a period, contradicting the premise that p is the smallest non-null period. □

Sets of periods and autocorrelations. Let $U \in \Sigma^n$. We denote the *set of all periods* of U by $P(U)$. We have that $P(U) \subseteq [0, n-1]$. The *autocorrelation* v of U is a representation of $P(U)$. It is a binary vector of length n such that: $\forall\, 0 \le i < n$, $v_i = 1$ iff $i \in P(U)$, and $v_i = 0$ otherwise. As v and $P(U)$ represent the same set, we use them interchangeably and write $P(U) = v$. We use both $i \in v$ and $v_i = 1$ to express that i is a period of a word U with autocorrelation v. We also write that *i is a period of v*. The smallest non-null period of U or of v is called its *basic period* and is denoted by $\pi(U)$ or $\pi(v)$.

We denote the concatenation of two binary strings s and t by $s \circ t$, and the k-fold concatenation of s with itself by s^k. So $\mathtt{10}^k \circ w$ is the string starting with 1, followed by k 0s, and ending with the string w.

Let $\Gamma(n) := \{v \in \{0,1\}^n \mid \exists U \in \Sigma^n : v = P(U)\}$ be the set of all autocorrelations of strings in Σ^n. We denote its cardinality by $\kappa(n)$. The autocorrelations in $\Gamma(n)$ can be partitioned according to their basic period; thus, for $0 \leq p < n$, we denote by $\Gamma(n,p)$ the subset of autocorrelations whose basic period is p, and by $\kappa(n,p)$ the cardinality of this set. The set inclusion defines a partial order on elements of $\Gamma(n)$. For $u, v \in \Gamma(n)$, we denote by $u \subseteq v$, resp. by $u \subset v$, the inclusion, resp. the strict inclusion, of u in v. We write $v \succ u$ if v covers u in the inclusion relationship, i.e., if $u \subset v$, and $u \subseteq y \subset v$ implies $y = u$.

1.2 Characterization of Correlations

In [6], Guibas and Odlyzko characterized the correlations of length n in terms of the Forward Propagation Rule (FPR), the Backward Propagation Rule (BPR), and by a recursive predicate Ξ. We review the main theorem and the definitions.

Theorem 2 (Characterization of Correlations [6]). *Let $v \in \{0,1\}^n$. The following statements are equivalent:*

1. *v is the correlation of a binary word*
2. *v is the correlation of a word over an alphabet of size ≥ 2*
3. *$v_0 = 1$ and v satisfies the Forward and Backward Propagation Rules*
4. *v satisfies the predicate Ξ.*

Definition 2. FPR, BPR, Predicate Ξ. Let $v \in \{0,1\}^n$.

FPR: v satisfies the FPR iff for all pairs (p,q) satisfying $0 \leq p < q < n$ and $v_p = v_q = 1$, it follows that $v_{p+i(q-p)} = 1$ for all $i = 2, \ldots, \lfloor \frac{n-p}{q-p} \rfloor$.

BPR: v satisfies the BPR iff for all pairs (p,q) satisfying $0 \leq p < q < 2p$, $v_p = v_q = 1$, and $v_{2p-q} = 0$, it follows that $v_{p-i(q-p)} = 0$ for all $i = 2, \ldots, \min(\lfloor \frac{p}{q-p} \rfloor, \lfloor \frac{n-p}{q-p} \rfloor)$.

Predicate Ξ: v satisfies Ξ iff v_0=1 and, if p is the basic period of v, one of the following conditions is satisfied:

Case a: $p \leq \lfloor \frac{n}{2} \rfloor$

Let $r := \mathrm{mod}(n,p)$, $q := p + r$ and w the suffix of v of length q. Then for all j in $[1, n-q]$ $v_j = 1$ if $j = ip$ for some i, and $v_j = 0$ otherwise; and the following conditions hold:

1. $r = 0$ or $w_p = 1$
2. if $\pi(w) < p$ then $\pi(w) + p > q + \gcd(\pi(w), p)$
3. w satisfies predicate Ξ.

Case b: $p > \lfloor \frac{n}{2} \rfloor$

We have: $\forall j : \ 1 \leq j < p, \ v_j = 0$. Let w be the suffix of v of length $n - p$, then w satisfies predicate Ξ.

Guibas and Odlyzko proved that verifying the predicate requires $O(n)$ time. Note that Ξ is recursive on the length of the binary vector. When v is tested, Ξ is recursively applied to a unique suffix of v denoted w (in case a, $|w| = p + r$; in case b, $|w| = n - p$). We call the corresponding w the *nested autocorrelation* of v. The following theorem is a consequence of the FPR and BPR, and of characterization (3) in Theorem 2 (see [6]).

Theorem 3. *Let v be a correlation of length n. Any substring $v_i \dots v_j$ of v with $0 \le i \le j < n$ such that $v_i = 1$ is a correlation of length $j - i + 1$.*

2 Irreducible Periods

We show that the period set of a word is in one-to-one correspondence with a smaller set which we call its associated *irreducible period set* (*IPS* for short).

A full period set contains redundancies since some periods are deducible from others as specified by the Forward Propagation Rule (FPR, see Section 1.2). For example with $n = 12$, in the period set $\{0, 7, 9, 11\}$, 11 can be obtained from 7 and 9 using the FPR ($11 = 9 + 1(9 - 7)$) and is the only deducible period. The IPS is thus $\{0, 7, 9\}$. In this section, we formally define the notion of IPS and we prove that the mapping R from $\Gamma(n)$ to $\Lambda(n)$, the set of all IPSs, is bijective. We also show how to compute the IPS from the period set, and conversely.

For every $n \in \mathbb{N}$, we define a function FC_n, the *Forward Closure*, from $2^{[0,n-1]}$ to $2^{[0,n-1]}$. Intuitively, FC_n repeatedly applies the FPR to all pairs of elements until closure is reached. Note that the order in which pairs of elements are considered does not matter, and therefore FC_n is well defined.

Definition 3 (Irreducible Period Set). Let $T \in \Gamma(n)$ be a period set. A subset $S := \{p_0, \dots, p_l\}$ of T is an associated irreducible period set (IPS) of T iff it satisfies both following conditions:

1. T is the forward closure of S, i.e., $\mathrm{FC}_n(S) = T$,
2. For all triples (h, i, j) satisfying $0 \le h < i < j \le l$ we have $\forall\, k \in \mathbb{N}^+ : p_j \neq p_i + k(p_i - p_h)$

Condition (2) expresses formally the fact that in an IPS no period can be obtained from smaller periods with the FPR. It is equivalent to saying that S is the *smallest* subset of T such that $\mathrm{FC}_n(S) = T$. In other words, S is an IPS of T if it is the intersection of all sets whose forward closure is T. From this, one can see that the associated IPS exists and is unique. Therefore, we can define a function R that maps a period set to its associated IPS. Now, we define $\Lambda(n) := R(\Gamma(n))$ and prove that the correspondence between period sets and IPSs is one-to-one.

Theorem 4. *$R : \Gamma(n) \to \Lambda(n),\ P \mapsto R(P)$ is bijective.*

Proof. By definition, R is surjective. To prove that R is injective we need to show that $R(P) = R(Q)$ implies $P = Q$. If $R(P) = R(Q)$ then $P = \mathrm{FC}_n(R(P)) = \mathrm{FC}_n(R(Q)) = Q$ by condition (1) of Definition 3. □

Algorithm 1: R

Input : Word length n, array P of periods in increasing order, size t of P
Output: Associated IPS $R(P)$ as an array I; **Variable**: S: a sorted set;

```
1  I[0] := P[0]; δ := n; i := 1; k := 1; S := ∅;
2  while ((i < t) and (δ > 1)) do
3      δ := P[i] − P[i − 1]; size := n − P[i − 1]; mul := ⌊size/δ⌋;
4      if P[i] ∉ S then
5          I[k] := P[i]; k := k + 1;
6          if mul = 2 then
7              if mod(size, δ) ≠ 0 then S.insert(P[i] + δ);
8          else if mul > 2 then S.insert(P[i − 1] + mul × δ); i := i + mul − 2;
9      i := i + 1;
10 return I;
```

By Theorem 4, R^{-1} exists; indeed, it is FC_n restricted to $\Lambda(n)$. Algorithm 1 is an efficient implementation of R. The next theorem claims that R runs in a time sublinear in the input size (which may be as large as $\Theta(n)$) because $|R(P)| = O(\log n)$ (We omit the proof and the algorithm R^{-1}.) This is achieved by exploiting the known structure of period sets; the algorithm does not need to examine the whole input array P (cf. line 8 of R).

Theorem 5. *For a given word length n and $P \in \Gamma(n)$, Algorithm 1 correctly computes $R(P)$ in $O(|R(P)| \log(|R(P)|))$ time.*

Proof. R considers the periods of P in increasing order and uses the sorted set S to store the forthcoming deducible periods. For each $P[i]$, R tests whether it is an irreducible period (line 4). If it is not, it is skipped; otherwise it is copied into I (line 5), and we are either in case (a) or (b) of Predicate Ξ. In case (b), no deducible periods are induced by $P[i]$, so nothing else is done. In case (a), we have $mul \geq 2$. If $mul = 2$ and $\mathrm{mod}(size, \delta) \neq 0$, the forward propagation generates only $P[i] + \delta$ which is inserted into S (lines 6 and 7). If $mul > 2$, Lemma 1 allows to skip the periods in the range $[P[i], P[i] + (mul - 2) \times \delta]$ and insert only $P[i-1] + mul \times \delta$, which is done on line 8. This proves the correctness.

We now prove that the running time is $O(|R(P)| \log |R(P)|)$. We claim that the while loop is executed at most $2 \cdot (R(P) - 1)$ times. Indeed, in each iteration, either an element is inserted into I and possibly into S, or nothing happens; the latter case arises only when the current $P[i]$ is in S. But at most $R(P) - 1$ elements are ever inserted into S and I, as after termination $|I| = |R(P)|$. Clearly, every operation in the loop takes constant time, except the operations on S, which take $O(\log |S|)$ time when S is implemented as a balanced tree. □

3 Structural Properties of $\Gamma(n)$ and $\Lambda(n)$

3.1 $\Gamma(n)$ Is a Lattice Under Inclusion

First, we prove that the intersection of two period sets is a period set.

Lemma 2. *If $u, v \in \Gamma(n)$, then $(u \cap v) \in \Gamma(n)$.*

Proof. Let $u, v \in \Gamma(n)$ and $w := u \cap v$. The results hold when $n = 1$. If $w = \{0\}$ we are done. Otherwise assume that for all $q < n, u', v' \in \Gamma(q)$ we have $(u' \cap v') \in \Gamma(q)$. Let p be the smallest common non-null period of u and v. So p is the smallest non-null period of w.

Case $p \leq \lfloor \frac{n}{2} \rfloor$: Let $i := \lfloor \frac{n}{p} \rfloor$. We have that multiples of p are periods of u and v: $\forall\ 1 \leq j \leq i;\ u_{p \cdot j} = v_{p \cdot j} = 1$ and so $w_{p \cdot j} = 1$. Moreover, we have $\forall\ 0 < k < n - p,\ k \neq jp :\ u_k \neq v_k$, otherwise p would not be the smallest common period of u and v. Hence, for all such k: $w_k = 0$. Consider the suffixes u', v', w' of length $n' := n - (i-1)p$ of u, v and w respectively. We know that $w = (\mathtt{10}^{p-1})^{i-1} \circ w'$, $u'_0 = v'_0 = u'_p = v'_p = 1$, and $w' = u' \cap v'$. As from Theorem 3, we know that $u', v' \in \Gamma(n')$, we have by induction that $w' \in \Gamma(n')$. Because w' satisfies the Theorem of Fine and Wilf, we have $\pi(w') + p > n' + \gcd(\pi(w'), p)$. Hence, w satisfies Predicate Ξ, i.e., $w \in \Gamma(n)$.

Case $p > \lfloor \frac{n}{2} \rfloor$: Let u', respectively v', be the suffix of length $n - p$ of u, resp. of v. By Theorem 3, u', v' are autocorrelations of size $n - p$. As $w = \mathtt{10}^{p-1} \circ (u' \cap v')$, by induction it fulfills Predicate Ξ. □

Lemma 3. *$(\Gamma(n), \subseteq)$ has a null element, $\mathtt{10}^{n-1}$, and a universal element, $\mathtt{1}^n$.*

Theorem 6. *$(\Gamma(n), \subseteq)$ is a lattice.*

Proof. From Lemma 2, we know that $\Gamma(n)$ is closed under intersection. Therefore, the meet $u \wedge v$ of $u, v \in \Gamma(n)$ is their intersection, and the join $u \vee v$ is the intersection of all elements containing both u and v. The existence of a universal element ensures that this intersection is not empty. □

3.2 $\Gamma(n)$ Does Not Satisfy the Jordan-Dedekind Condition

We demonstrate that $\Gamma(n)$ does not satisfy the Jordan-Dedekind condition, implying that it is neither modular, distributive, nor a matroid. The next lemma proves the existence of a specific maximal chain[1] between $\mathtt{1}^n$ and $\mathtt{10}^{n-1}$ in $\Gamma(n)$.

Lemma 4. *Let $n \in \mathbb{N}$ and $p := \lfloor \frac{n}{2} \rfloor + 1$. The following chain exists in $\Gamma(n)$:*

$$\mathtt{1}^n \succ \mathtt{10}^{p-1}\mathtt{1}^{n-p} \tag{1}$$

$$\forall\ p \geq i \geq n-2\ :\ \mathtt{10}^{i-1}\mathtt{1}^{n-i} \succ \mathtt{10}^{i}\mathtt{1}^{n-i-1} \tag{2}$$

$$\mathtt{10}^{n-2}\mathtt{1} \succ \mathtt{10}^{n-1} \tag{3}$$

Moreover, this chain is maximal and has length $\lceil \frac{n}{2} \rceil$.

[1] In a partially ordered set or *poset*, a *chain* is defined as a subset of completely ordered elements, an *antichain* as a subset in which any two elements are uncomparable. The length of a chain is its number of elements minus one.

Proof. We prove (1). Obviously, $1^n \supset 10^{p-1}1^{n-p}$. We must show that: if $1^n \supset x \supseteq 10^{p-1}1^{n-p}$ then $x = 10^{p-1}1^{n-p}$. Assume that such an x exists and is different from $10^{p-1}1^{n-p}$. Then $0 < \pi(x) < p$ and $x_{\pi(x)} = 1$. By Lemma 1, we have $\forall\, j < n - \pi(x) + 2$, $x_j = 0$ iff $\pi(x) \nmid j$. Thus, for some $p \leq k < n$, $x_k = 0$ and $x \not\supseteq 10^{p-1}1^{n-p}$, which is a contradiction.

The autocorrelations involved in (2) and (3) exist by Predicate Ξ and only differ from each other by one period. This implies (2) and (3) and proves that the chain is maximal. By counting the links of the chain, one gets $n-p+1 = \lceil \frac{n}{2} \rceil$. □

With $p := \lfloor \frac{n}{2} \rfloor + 1$ as above, consider $\Gamma(n,p)$ and its associated sub-lattice in $\Gamma(n)$. From Predicate Ξ, we have that $\Gamma(n,p) = \{10^{p-1}\} \circ \Gamma(n-p)$. So the structure of the sub-lattice defined by $\Gamma(n,p)$ is exactly the one of the lattice of $\Gamma(n-p)$. Using the previous lemma, we deduce the existence of an induced maximal chain between $10^{p-1}1^{n-p}$ and $10^{p-1}10^{n-p-1}$ in $\Gamma(n)$. Combining this with Equation 1 and $10^{p-1}10^{n-p-1} \succ 10^{n-1}$, we obtain another maximal chain between 1^n and 10^{n-1} in $\Gamma(n)$. This proves the following lemma.

Lemma 5. *Let $n > 8$ and $p := \lfloor \frac{n}{2} \rfloor + 1$ be integers. The chain going from 1^n to $10^{p-1}1^{n-p}$, from there to $10^{p-1}10^{n-p-1}$ through the induced maximal chain over $\Gamma(n,p)$, and then to 10^{n-1} is a maximal chain of $\Gamma(n)$. Its length is $\lceil \frac{\lceil \frac{n}{2} \rceil - 1}{2} \rceil + 2$.*

Hand inspection for $n := 1, \ldots, 6$ shows that $\Gamma(n)$ satisfies the Jordan-Dedekind condition, i.e., all maximal chains between the same elements have the same length. We now demonstrate it is not the case when $n > 6$.

Theorem 7. *For $n > 6$, $\Gamma(n)$ does not satisfy the Jordan-Dedekind condition.*

Proof. From lemmas 4 and 5, we obtain the existence between 1^n and 10^{n-1} of two maximal chains of lengths $\lceil \frac{n}{2} \rceil$ and $\lceil \frac{\lceil \frac{n}{2} \rceil - 1}{2} \rceil + 2$. Clearly, for $n > 8$ these are different. Moreover, hand inspection of $\Gamma(7)$ and $\Gamma(8)$ shows that they also do not fulfill the Jordan-Dedekind condition. □

3.3 The Poset $(\Lambda(n), \subseteq)$ Satisfies the Jordan-Dedekind Condition

For $n \geq 3$, $(\Lambda(n), \subseteq)$ is not a lattice ($\{0,1\}$ and $\{0,2\}$ never have a join). On the other hand, in contrast to $\Gamma(n)$, we have the stronger result that any subset of an IPS containing 0 is an IPS.

Lemma 6. *Let $R \in \Lambda(n)$ and let $\{0\} \subset Q \subset R$, then $Q \in \Lambda(n)$.*

Proof. Let $P := \mathrm{FC}_n(R) \in \Gamma(n)$. We must show that $P' := \mathrm{FC}_n(Q) \in \Gamma(n)$, and that no element of Q is deducible from others by the FPR. The latter property follows from the minimality of R. To show $P' \in \Gamma(n)$, we only need to consider the special case where $R = Q \dot{\cup} \{t\}$, i.e., where Q contains exactly one element less than R. The general case follows by repeated application of the special case.

For a contradiction, assume $P' \notin \Gamma(n)$. Since P' satisfies the FPR, it must violate the BPR (see Characterization (3) of Theorem 2). So let $0 < p < q < n$ with $\delta := q - p$ such that $p - \delta \notin P'$, but $p - i\delta \in P'$ for some $i \in \{2, \ldots, \min(\lfloor \frac{p}{q-p} \rfloor, \lfloor \frac{n-p}{q-p} \rfloor)\}$. Since P does satisfy the BPR, we must have that $p - \delta \in P$, and this must be a result of adding t to Q and propagating it. From this, we conclude that one of the supposedly non-deducible elements of Q, and hence of R, is in fact deducible from t. So R is not an IPS, a contradiction. □

Theorem 8. *The set $\Lambda(n)$ of all Irreducible Period Sets is partially ordered and satisfies the Jordan-Dedekind condition with respect to set inclusion.*

Proof. Clearly, set inclusion induces a partial order on $\Lambda(n)$. From Lemma 6, for all pairs $P, Q \in \Lambda(n)$: $P \succ Q$ iff $P = Q \cup \{q\}$ for some q in $[1, n-1]$. Thus, any two maximal chains between the same element have the same length. □

As a corollary of Lemma 6, the intersection of two IPSs is an IPS, but the intersections of two IPSs is not the IPS of the intersection of their respective period sets. Neither $\Gamma(n)$ nor $\Lambda(n)$ are closed under union. The union of two IPSs may recursively violate Theorem 1 several times, as in the following example: $u := \{0, 5, 7\}$, $v := \{0, 5, 8, 9\}$, $u \cup v = \{0, 5, 7, 8, 9\}$ ($(7, 8)$ require 6 in the suffix of length 5, and $(5, 6)$ require 1 in the whole $u \cup v$).

4 Enumeration of All Autocorrelations of Length n

In this section, we present the first enumeration algorithm for string autocorrelations of length n. A brute force algorithm is to apply Predicate Ξ to each of the 2^n possible binary vectors and retain those that satisfy Ξ. This is exponential in n and not practical. The recursive structure of Ξ permits the use of Ξ as the basis of a dynamic programming algorithm that efficiently computes $\Gamma(n)$ from $\Gamma(m, p)$ with $m < 2n/3$ and $1 \le p \le m$. $\Gamma(n, 1) = \{1^n\}$ and $\Gamma(n, n) = \{10^{n-1}\}$ for all n. Below is the algorithm to compute $\Gamma(n, p)$ for $n \ge 3$ and $2 \le p \le (n-1)$. We assume that all necessary $\Gamma(m, p)$ with $m < 2n/3$ have already been computed.

Case (a) [$2 \le p \le \frac{n}{2}$]: Let $r' := n \bmod p$ and $r := r' + p$. Then $p \le r < 2p$, and there are two sub-cases. In each of them, $\Gamma(n, p)$ can be constructed from a subset of $\Gamma(r)$. Let $s_{n,p} := (10^{p-1})^{\lfloor n/p \rfloor - 1}$; every correlation in $\Gamma(n, p)$ is of the form $s_{n,p} \circ w$ with $w \in \Gamma(r)$ chosen as follows.

1. Case $r = p$:

$$\Gamma(n, p) = \{s_{n,p} \circ w \mid w \in \Gamma(r, p');\ r' + \gcd(p, p') < p' < p\} \tag{4}$$

2. Case $p < r < 2p$:

$$\Gamma(n, p) = \{s_{n,p} \circ w \mid w \in \Gamma(r, p)\} \tag{5}$$
$$\dot{\bigcup} \{s_{n,p} \circ w \mid w \in \Gamma(r, p');\ r' + \gcd(p, p') < p' < p;\ \ w_p = 1\}$$

In (4) and (5) : $(r' + \gcd(p, p') < p' < p) \ \Rightarrow p' \nmid p$.

Case (b) [$\frac{n}{2} < p \le (n-1)$]: $\Gamma(n, p)$ is constructed from $\Gamma(n-p)$.

$$\Gamma(n, p) = \{10^{p-1} \circ w \mid w \in \Gamma(n-p)\} \tag{6}$$

Proof (Correctness). Comparison with Ξ reveals that every element that is included in $\Gamma(n, p)$ according to each of (4), (5), or (6) fulfills Ξ. (Case (a) of Ξ has been further subdivided into $r = p$ and $p < r < 2p$.) It remains to be shown that every vector satisfying Ξ is included in the appropriate $\Gamma(n, p)$. If this is not the case, let v be a vector of minimal length n that is an autocorrelation, but that is not included in $\Gamma(n, p)$ where $p = \pi(v)$. The only way this could happen would be if the r-suffix of v were already not contained in its appropriate $\Gamma(r, p')$. But this would contradict the minimality of n. □

Improvements. Two improvements increase the efficiency and allow computation up to $n = 450$.

1. For given values of n and p, all autocorrelations in $\Gamma(n,p)$ have the same prefix. The prefix length is p for $p > \frac{n}{2}$ and $p\,(\lfloor n/p \rfloor - 1)$ for $p \leq \frac{n}{2}$. This prefix is immediately available, and need not be stored explicitly.
2. In case (a), $\Gamma(n,p)$ is obtained from autocorrelations $w \in \Gamma(r)$ with $r \geq p$. By Lemma 1, such w must satisfy $\pi(w) > (n \bmod p)$, and therefore it is possible to construct $\Gamma(n,p)$ from the sets $\Gamma(s)$ with $s < p$. Hence, to obtain $\Gamma(n,p)$, in both cases (a) and (b), only the sets $\Gamma(m,p')$ with $m \leq \lfloor \frac{n}{2} \rfloor$, $1 \leq p' \leq m$ are needed. For example, to compute $\Gamma(200)$, we only need to know $\Gamma(1), \ldots, \Gamma(100)$ and their respective subsets, but not $\Gamma(101), \ldots, \Gamma(133)$.

5 Bounds on the Number of Autocorrelations

In this section, we investigate how the number $\kappa(n)$ of different autocorrelations of length n grows with n. From Theorem 2, we know that $\kappa(n)$ is independent of the alphabet size. In [6], it is shown that as $n \to \infty$,

$$\frac{1}{2\ln 2} + o(1) \leq \frac{\ln \kappa_n}{(\ln n)^2} \leq \frac{1}{2\ln(3/2)} + o(1). \tag{7}$$

As shown in Figure 1, these bounds are rather loose. In fact, for small n, the actual value of $\kappa(n)$ is below its asymptotic lower bound. While we conjecture that $\lim_{n\to\infty} \frac{\ln \kappa_n}{(\ln n)^2} = \frac{1}{2\ln 2}$, it remains an open problem to derive a tight upper bound and prove this conjecture. Our contribution is that a good lower bound for κ_n is closely related to the number of binary partitions of an integer. Both improved bounds we derive from this relationship are also shown in Figure 1.

We have $\kappa_0 = 1$, $\kappa_1 = 1$, and $\kappa_2 = 2$. Considering only the correlations given by case (b) of Predicate Ξ, we have $\kappa_n \geq \sum_{n/2<p\leq n} \kappa_{n-p} = \sum_{i=0}^{\lceil n/2 \rceil - 1} \kappa_i$. We define $L_0 := 1$, $L_1 := 1$, and, for $n \geq 2$, $L_n := \sum_{i=0}^{\lceil n/2 \rceil - 1} L_i$. By induction, $L_n \leq \kappa_n$ for all $n \geq 0$. From the definition of L_n, we deduce that for $n \geq 2$, $L_n = L_{n-1}$ for n even, and $L_n = L_{n-2} + L_{\frac{n-1}{2}}$ for n odd.

Now we consider a related sequence: the number of binary partitions B_n of an integer $n \geq 0$, i.e., the number of ways to write n as a sum of powers of 2 where the order of summands does not matter. For example, 6 can be written as such a sum in 6 different ways: 4+2, 4+1+1, 2+2+2, 2+2+1+1, 2+1+1+1+1, 1+1+1+1+1+1. Therefore $B_6 = 6$. By convention, $B_0 = 1$; furthermore $B_1 = 1$. Binary partitions have been extensively studied; for example, see [3,5]. For $n \geq 2$, they satisfy the recursion $B_n = B_{n-2} + B_{\frac{n}{2}}$ for n even and $B_n = B_{n-1}$ for n odd. The following lemma states the close relation between the lower bound L_n for $\kappa(n)$ and the number of binary partitions B_n.

Lemma 7. *For $n \geq 1$, $L_n = 1/2 \cdot B_{n+1}$.*

Proof. The proof is by induction. For $n = 1$, we have $L_1 = 1 = 1/2 \cdot B_2$. If $n \geq 2$ is even, $L_n = L_{n-1} = \frac{1}{2} \cdot B_{(n-1)+1} = \frac{1}{2} \cdot B_{n+1}$, as $(n+1)$ is then odd. If $n \geq 3$ is odd, $L_n = L_{n-2} + L_{\frac{n-1}{2}} = \frac{1}{2}\left(B_{n-1} + B_{\frac{n+1}{2}}\right) = \frac{1}{2} \cdot B_{n+1}$, by the recursion for B_{n+1} for even $(n+1)$. □

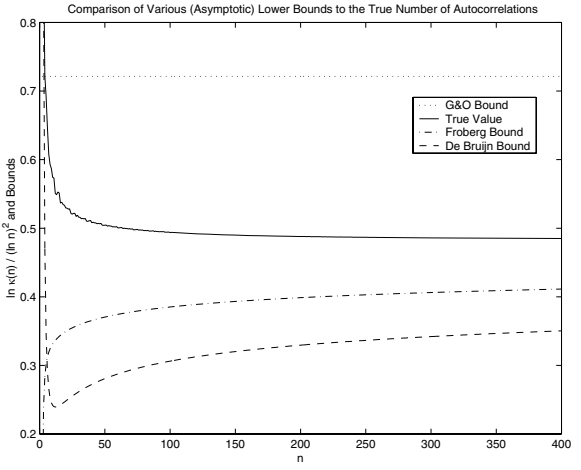


Fig. 1. True values of $\ln \kappa_n/(\ln n)^2$ for $n \leq 400$, compared to Guibas & Odlyzko's (G&O) asymptotic lower bound, the improved asymptotic bound from Theorem 9 (ii) derived from DeBruijn's results, and the non-asymptotic lower bound from Theorem 9 (i) based on Fröberg's work. Both of these bounds converge to the G&O asymptotic value of $1/(2\ln 2)$ for $n \to \infty$. The upper bound of G&O, corresponding to the line $y = 1/(2\ln(3/2)) \approx 1.23$, is not visible on the figure.

Fröberg [5] and De Bruijn [3] give some bounds on B_n. Combining Lemma 7, Fröberg's and De Bruijn's results allows us to derive good lower bounds on $\kappa(n)$ in the next Theorem (The proof is omitted).

Theorem 9 (Lower Bounds on $\kappa(n)$). *Define*

$$F(n) := \sum_{k=0}^{\infty} \frac{n^k}{2^{\frac{k(k+1)}{2}} \cdot k!}. \tag{8}$$

i/ For all $n \geq 1$, $\kappa_n \geq 0.31861 \cdot F(n+1)$. ii/ Asymptotically (with approximated constants),

$$\frac{\ln \kappa_n}{(\ln n)^2} \geq \frac{1}{2\ln 2}\left(1 - \frac{\ln\ln n}{\ln n}\right)^2 + \frac{0.4139}{\ln n} - \frac{1.47123\,\ln\ln n}{(\ln n)^2} + O\left(\frac{1}{(\ln n)^2}\right).$$

6 Computing the Size of Populations

The correlation of a string depends on its self-overlapping structure, but is not directly related to its characters. Hence, different strings share the same correlation. For instance over the alphabet $\{a, b\}$, take *abbabba* and *babbabb*. The *population* of a correlation v is the set of strings over Σ whose correlation is v.

We wish to compute the *size of the population* of a given correlation, and by extension of all correlations.

In [6], Guibas and Odlyzko exhibit a recurrence linking the population sizes of a correlation and of its nested correlation. Here, we exhibit another recurrence which links the population size of an autocorrelation v to the population sizes of the autocorrelations it is included in. The recurrence depends on the *number of free characters* (nfc for short) of v, to be defined next.

Definition 4 (Number of Free Characters). The nfc of a correlation v is the maximum number of positions in a string U with $P(U) = v$ that are not determined by the periods.

To illustrate this definition, note that a correlation represents a set of equalities between the characters of a string. For example, take $v := 100001001 \in \Gamma(9)$. A string $U = u_0 \dots u_8$ with $P(U) = v$ must satisfy the following set of equations: $\{u_0 = u_3 = u_5 = u_8, u_1 = u_6, u_2 = u_7\}$. Thus we can write any word U as $u_0u_1u_2u_0u_4u_0u_1u_2u_0$ for some $u_0, u_1, u_2, u_4 \in \Sigma$. So the nfc of v is 4.

The nfc is independent of Σ and can be computed from v alone. Given a correlation v and its length n, Algorithm 2 (NFC), computes the nfc of v. NFC follows the recursive structure of Predicate Ξ and requires $\Theta(n)$ time.

Algorithm 2: *NFC*

Input: $n \in \mathbb{N}$, $v \in \Gamma(n)$; **Output**: the number of free characters of v;

```
1 i := 1;while (i < n) and (v_i ≠ 1) do i := i + 1;// search for the basic period ;
2 if i = n then return n;// no basic period ;
3 if i = 1 then return 1;
4 if (i ≤ ⌊n/2⌋) then return NFC(i + mod(n, i), v[n − i − mod(n, i)..n − 1]);
5 else return 2 × i − n + NFC(n − i, v[i..n − 1]);
```

We now state our recurrence on the population sizes.

Theorem 10. *Let $n \in \mathbb{N}$ and let v_k be the k-th $(k = 1, \dots, \kappa(n))$ autocorrelation of $\Gamma(n)$. Let ρ_k denote the number of free characters of v_k, and N_k be its population size. We have:*

$$N_k = \sigma^{\rho_k} - \sum_{j : v_k \subset v_j} N_j.$$

Proof. For any word U with $P(U) = v_k$ there are ρ_k free positions. For each of the σ^{ρ_k} combinations of ρ_k characters from Σ, we construct a word V satisfying the character equalities associated with v_k, and have $v_k \subseteq P(V)$. We do not necessarily have $v_k = P(V)$, because V may in fact satisfy additional character equalities. Conversely, every word V with $v_k \subseteq P(V)$ is obtained in this way. Therefore

$$\sigma^{\rho_k} = \sum_{j : v_k \subseteq v_j} N_j = N_k + \sum_{j : v_k \subset v_j} N_j,$$

which proves the theorem. □

Acknowledgments. We thank D. Bryant, the groups of S. Schbath at INRA Jouy en Josas, and of Ph. Flajolet at INRIA Rocquencourt for helpful discussions. E.R. is supported by the CNRS, part of this work has been done while working at the DKFZ, in Heidelberg, Germany. S. R. is grateful to LIRMM for a travel grant.

References

1. C. Choffrut and J. Karhumäki. Combinatorics of words. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 329–438. Springer-Verlag, 1997.
2. M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, 1994.
3. N. G. DeBruijn. On Mahler's partition problem. *Proc. Akad. Wet. Amsterdam*, 51:659–669, 1948.
4. N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proc. Amer. Math. Soc.*, 16:109–114, 1965.
5. C.-E. Fröberg. Accurate estimation of the number of binary partitions. *BIT*, 17:386–391, 1977.
6. L. J. Guibas and A. M. Odlyzko. Periods in strings. *Journal of Combinatorial Theory,* Series A, 30:19–42, 1981.
7. M. Lothaire. *Algebraic Combinatorics on Words.* in preparation, 1999. URL: http://www-igm.univ-mlv.fr/~berstel/Lothaire/index.html.
8. S. Rahmann and E. Rivals. Exact and Efficient Computation of the Expected Number of Missing and Common Words in Random Texts. In R. Giancarlo and D. Sankoff, editors, *Proc. of the 11th Symposium on Combinatorial Pattern Matching*, number 1848 in LNCS, pages 375–387, Montréal, Canada, 2000. Springer-Verlag, Berlin.
9. R. Sedgewick and P. Flajolet. *Analysis of Algorithms.* Addison-Wesley, Reading, MA, 1996.

Minimal Tail-Biting Trellises for Certain Cyclic Block Codes Are Easy to Construct

Priti Shankar, P.N.A. Kumar, Harmeet Singh, and B.S. Rajan

Indian Institute of Science, Bangalore, 560012.
{priti,pnakumar}@csa.iisc.ernet.in,
harmeet@sasken.com,bsrajan@ece.iisc.ernet.in

Abstract. We give simple algorithms for the construction of generator matrices for minimal tail-biting trellises for a powerful and practical subclass of the linear cyclic codes, from which the combinatorial representation in the form of a graph can be obtained by standard procedures.

Keywords: linear block codes, cyclic codes, Reed-Solomon codes, tail-biting trellises.

1 Introduction

Trellis descriptions of block codes [1,15,8,10,3,6,9] are *combinatorial* descriptions, as opposed to the traditional *algebraic* descriptions of block codes. A minimal *conventional* trellis for a linear block code is just the transition graph for the minimal finite state automaton which accepts the language consisting of the set of all codewords. With such a description, the decoding problem reduces to finding a cheapest accepting path in such an automaton (where transitions are assigned costs based on a channel model.) However, trellises for many useful block codes are often too large to be of practical value. Of immense interest therefore, are *tail-biting* trellises for block codes, recently introduced in [2], which have reduced state complexity. The strings accepted by a finite state machine represented by a trellis are all of the same length, that is the *block length* of the code. Coding theorists therefore attach to all states that can be reached by strings of the same length l, a *time index* l. Conventional trellises use a linear time index, whereas tail-biting trellises use a circular time index. It has been observed [14] that the maximum state cardinality of a tail-biting trellis at any time index can drop to the square root of the maximum state cardinality (over all time indices) of a conventional trellis for the code, thus increasing the potential practical applications of trellis representations for block codes. In this paper, we show that finding a minimal tail-biting trellis corresponds to picking basis vectors of the vector space defining the code in a particular way, and using the selected vectors to build up the trellis. We then show that for various subclasses of cyclic codes, obtaining vectors that span the space and that also yield minimal tail-biting trellises is easy. Section 2 presents the background. Section 3 presents our results on cyclic codes, and Section 4 gives results for Reed-Solomon codes.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 627–638, 2001.

2 Background

We give a very brief background on subclasses of block codes called linear codes. Readers are referred to the classic text [7].

Let $GF(q)$ denote the field with q elements. It is customary to define linear codes algebraically as follows:

Definition 1. *A linear block code C of length n over a field $GF(q)$ is a k-dimensional subspace of an n-dimensional vector space over the field $GF(q)$ (such a code is called an (n, k) code.)*

The most common algebraic representation of a linear block code is the generator matrix G. A $k \times n$ matrix G where the rows of G are linearly independent and which generate the subspace corresponding to C is called a *generator matrix* for C. Figure 1 shows a generator matrix for a $(4, 2)$ linear code over $GF(2)$, consisting of the four codewords in the set {0000,0110,1001,1111}. A *cyclic* linear

$$\mathbf{G} = \begin{bmatrix} 0\ 1\ 1\ 0 \\ 1\ 0\ 0\ 1 \end{bmatrix}$$

Fig. 1. Generator matrix for a $(4, 2)$ linear binary code

block code satisfies the additional property that any cyclic shift of a codeword is also a codeword. (The code of the example above is linear but not cyclic.) Bose-Chaudhari-Hocquengham(BCH) codes and Reed-Solomon codes are the best known cyclic codes which have many practical applications. Cyclic (n, k) codes are generated by k multiples modulo $x^n - 1$ of a polynomial $g(x)$ of degree $n-k$, that divides $x^n - 1$. A codeword corresponds to the coefficients of the polynomial. The polynomial $g(x)$ is chosen to be monic and has degree $n-k$. The codewords corresponding to the multiples $g(x), xg(x), \ldots, x^{k-1}g(x)$ form a basis for the subspace that defines the code. The *parity check polynomial* $h(x)$ (of degree k) is defined as $h(x) = (x^n - 1)/g(x)$. For BCH codes, $g(x)$ has coefficients in a *ground field* $GF(q)$ and roots in an *extension field* $GF(q^m)$. For Reed-Solomon codes, the coefficients and roots of $g(x)$ are in the same field. An example of a BCH code is a binary (7,4) Hamming code. This is generated by $g(x) = x^3 + x + 1$ and has the generator matrix shown in Figure 2. The polynomial $g(x)$ for the (7,4) Hamming code above, has as roots, $\alpha, \alpha^2, \alpha^4$ where α is a primitive element of the field $GF(2^3)$. The polynomial $g(x)$ itself has coefficients in $GF(2)$. The parity check polynomial for this code is $h(x) = x^4 + x^2 + x + 1$ The polynomial $h(x)$ has as roots, all the remaining powers of α, namely, $\alpha^3, \alpha^5, \alpha^6$, and 1. Thus $g(x)$ and $h(x)$ between them have as roots all the non zero elements of the cyclic multiplicative group of the field $GF(2^3)$. These are the seven roots of $x^7 - 1$.

A general block code also has a *combinatorial* description in the form of a *trellis*. We borrow from Kschischang et al. [6] the definition of a trellis for a block code.

$$\mathbf{G} = \begin{bmatrix} 1\,1\,0\,1\,0\,0\,0 \\ 0\,1\,1\,0\,1\,0\,0 \\ 0\,0\,1\,1\,0\,1\,0 \\ 0\,0\,0\,1\,1\,0\,1 \end{bmatrix}$$

Fig. 2. Generator matrix for the $(7, 4)$ binary Hamming code

Definition 2. *A trellis for a block code C of length n, is an edge labeled directed graph with edge labels drawn from a set A, with a distinguished* root *vertex s, having in-degree* 0 *and a distinguished* goal *vertex f having out-degree* 0*, with the following properties:*

1. *All vertices can be reached from the root.*
2. *The goal can be reached from all vertices.*
3. *The number of edges traversed in passing from the root to the goal along any path is n.*
4. *The set of n-tuples obtained by "reading off" the edge labels encountered in traversing all paths from the root to the goal is C.*

The length of a path (in edges) from the root to any vertex is unique and is sometimes called the *time index* of the vertex. It is well known that minimal trellises for linear block codes are unique [10] and constructable from a generator matrix for the code [6]. In contrast, minimal trellises for non-linear codes are, in general, neither unique, nor deterministic [6]. Figure 3 shows a trellis for the linear code in Figure 1. Figure 4 shows the minimal conventional trellis for the

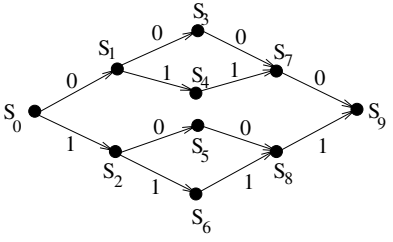


Fig. 3. A trellis for the linear block code of Figure 1 with $S_0 = s$ and $S_9 = f$

Hamming code of Figure 2. Minimal trellises for linear codes are transition graphs for bideterministic automata. The trellises are said to be *biproper*. Biproper trellises minimize a wide variety of structural complexity measures. McEliece [9] has defined a measure of Viterbi decoding complexity in terms of the number of edges and vertices of a trellis, and has shown that the biproper trellis is

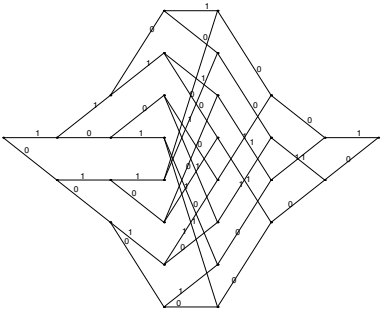


Fig. 4. A conventional trellis for the Hamming code of Figure 2

the "best" trellis using this measure, as well as other measures based on the maximum number of states at any time index, and the total number of states.

We briefly mention the algorithm given in [6] for constructing a minimal trellis from a generator matrix(in a specified form) for the code. An important component of the algorithm is the *trellis product* construction, whereby a trellis for a "sum" code can be obtained as a product of component trellises. The set of vertices of the product trellis at each time index, is just the Cartesian product of the vertices of the component trellis. If we define the i^{th} section as the set of edges connecting the vertices at time index i to those at time index $i+1$, then the edge count in the i^{th} section is the product of the edge counts in the i^{th} section of the individual trellises. Before the product is constructed we put the matrix in *trellis oriented form* described now. Given a non zero codeword $C = (c_1, c_2, \ldots c_n)$, $start(C)$ is the smallest integer i such that c_i is non zero. Also $end(C)$ is the largest integer for which c_i is nonzero. The *linear span* of C is $[start(C), end(C)]$. By convention the span of the all 0 codeword $\mathbf{0}$ is the empty span [ ]. The minimal trellis for the binary $(n, 1)$ code generated by a nonzero codeword with span $[a, b]$ is constructed as follows. There is only one path up to $a-1$ from index 0, and from b to n. From $a-1$ there are 2 outgoing branches diverging(corresponding to the 2 multiples of the codeword), and from $b-1$ to b, there are 2 branches converging. For a code over $GF(q)$ there will be q outgoing branches and q converging branches. It is easy to see that this is the minimal trellis for the 1-dimensional code, and is called the *elementary* trellis corresponding to the codeword. To generate the minimal trellis for C we first put the trellis into *trellis oriented form*, where for every pair of rows, with spans $[a_1, b_1], [a_2, b_2], a_1 \neq b_1$ and $a_2 \neq b_2$. We then construct individual trellises for the k 1-dimensional codes as described above, and then form the trellis product. Conversion of a generator matrix into trellis oriented form requires a sequence of operations similar to Gaussian elimination, applied twice. In the first phase, we apply the method to ensure that each row in the matrix starts its first nonzero entry at a time index one higher than the previous row. In the second phase we ensure that no two rows have their last nonzero entry at the same time index.

The complexity of the algorithm is $O(k^2n + s)$ for an (n, k) linear code whose minimal trellis has s states. We see that the generator matrices displayed earlier are already in trellis oriented form. The elementary trellises for the two rows of the generator matrix in Figure 1 are shown in Figure 5 below. The product of these two elementary trellises yields the trellis in Figure 3.

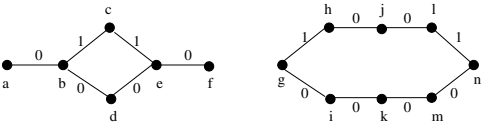


Fig. 5. Elementary trellises for the rows of the generator matrix in Figure 1

A *tail-biting trellis* is defined on a circular time axis of length n which is usually identified with $Z_n = \{0, 1, 2, \ldots n-1\}$, the integers modulo n. All arithmetic operations on indices are performed modulo n. For each time index $j \in Z_n$ there is a finite state space S_j. All edges of the tail-biting trellis are between S_j and $S_{j+1(modn)}$, and as in the conventional case, are labeled with elements from A. One can think of a tail-biting trellis as defined on a sequential time axis with

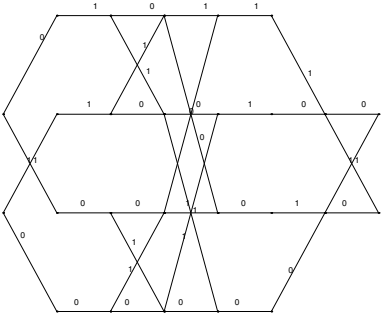


Fig. 6. A tail-biting trellis for the Hamming code of Figure 2

$S_0 = S_n$ and by restricting valid paths to those that begin and end at the same state. Figure 6 shows a tail-biting trellis for the Hamming code of Figure 2, with $|S_0| = |S_n| = 2$. One can also regard a conventional trellis as a tail-biting trellis with $|S_0| = |S_n| = 1$.

The *state cardinality profile* of a tail-biting trellis is the sequence $(|S_0|, |S_1|, \ldots |S_{n-1}|)$. For example, the state cardinality profile for the tail-biting trellis for the (7,4) Hamming code in Figure 6 is (2,4,4,4,4,4,2). The *maximum state cardinality* of a trellis is defined as $S_{max} = max\{|S_0|, |S_1|, \ldots |S_{n-1}|\}$. The minimum state

cardinality, S_{min} can similarly be defined. It is well known that for conventional trellises representing linear block codes, there is a unique minimal trellis for a given linear block code. However this is not true in general for tail-biting trellises. There are three notions of minimality that have been suggested for tail-biting trellises. In [2] a trellis is called μ-minimal if S_{max} is minimized over all possible permutations of the time index, and choices of generator matrices. A second definition is π-minimality, also defined in [2] where the goal is to minimize the product of all state space sizes over all permutations of the time indices and choices of generator matrices. Kotter and Vardy [5] define weaker notions of minimality. In their definitions a coordinate ordering is considered to be fixed. One definition is that of Θ-minimality. A trellis T is said to be smaller than or equal to another trellis T' denoted by $T \leq_\Theta T'$ if $|S_i| \leq |S'_i|$ for all $i \in \{0, 1, \ldots n-1\}$. If at least one of the indices has a strict inequality in the set of inequalities above, then $T < T'$. A second ordering is on the basis of the product of all state space sizes, as defined earlier, except that permutations of the time index are not considered.

We end this section by stating a few bounds. All of these are found in the excellent survey on trellises for block codes [13] and in [2]. Let C be an (n, k) linear block code over the field $GF(q)$.

1. For a conventional trellis, $S_{max} \leq min(q^k, q^{n-k})$
2. For a tail-biting trellis the product complexity, $\pi \geq q^{(d-1)k}$, where d is the minimum of the Hamming distances between all pairs of codewords. This will be referred to as the *total span* lower bound.
3. For a tail-biting trellis $S_{max} \geq q^{\lceil (d-1)k/n \rceil}$
4. If S_{mid} is the minimum possible state-space size of a conventional trellis for a code C at its midpoint under any permutation of its time index, then $S_{max} \geq \sqrt{S_{mid}}$. This is referred to as the *square root* lower bound.

3 An Algorithm for the Construction of Minimal Tail-Biting Trellises for Cyclic Codes

We first give another definition of minimality. Our definition is for a fixed coordinate ordering. We say that trellis T is Σ-minimal if $\sum_i |S_i| \leq \sum_i |S_i|'$ for any other trellis T'. This is a natural definition from the point of view of automata theory. However, we must recall that we are not dealing with conventional automata, but with automata having multiple start and multiple final states, and with a restricted definition of acceptance. We have observed that subject to minimization of the product of all state space sizes, the minimization of this quantity also minimizes S_{max}. Also this definition seems to favour "flat" trellises over others with the same product space size. Flat minimal trellises, in fact achieve the square root lower bound [2]. Another definition that favours flat trellises is Δ-minimality. We say that a tail-biting trellis is Δ-minimal for a given product size if $log(S_{max}) - log(S_{min})$ is minimal, where the logarithm is to the base q if the code symbols are from $GF(q)$. The decoding complexity in a conventional

trellis is closely related to the number of edges and nodes of the trellis. This is also true for tail-biting trellises [11]. It is thus of interest to minimize the Σ-size. We show that in the cases considered here, Δ-minimality implies Σ-minimality.

We now define a *circular span* of a codeword. Let $C = (c_1, c_2 \ldots c_n)$ be a codeword such that c_i and c_j are non-zero, $i < j$ and all components between i and j are zero. Then $[j, i]$ is said to be a *circular span* of the codeword. Note that while the linear span of a codeword is unique, a circular span is not, as it depends on the consecutive run of zeros chosen. Given a codeword and a circular span $[j, i]$, there is a unique elementary trellis corresponding to it. If the code symbols are from $GF(q)$, then the trellis has q states from index 0 to index $i-1$, one state from index i to index j and q states from index $j+1$ to index n. If the start states are numbered from 1 to q and final states likewise, only i to i paths are codewords. Shany and Beery[12] have shown that any linear tail-biting trellis can be constructed from a generator matrix whose rows can be partitioned into two sets, those which are taken to have linear span, and those taken to have circular span. The tail-biting trellis is then formed as a product of the elementary trellises corresponding to these rows. Thus the problem of constructing a minimal tail-biting trellis is then reduced to finding a basis for the subspace constituting the code, and a choice of spans, such that the corresponding elementary trellises yield a product that corresponds to a minimal tail-biting trellis.

It is convenient to introduce some notation at this point. A state complexity profile of the form $(q^l, q^l, \ldots i_1$ times, $q^j, q^j, \ldots i_2$ times, $q^m, q^m, \ldots i_3$ times) is represented as $((q^l)^{i_1}, (q^j)^{i_2}, (q^m)^{i_3})$. Let c be a codeword with linear span $[i, j]$. Then the elementary conventional trellis for the codeword has a profile of the form $(1^{i+1} q^{j-i} 1^{n-j-1})$.We abbreviate this as $([1, q, 1], [i+1, j])$, where indices in the trellis begin at 0, as the state cardinality remains constant at q in the interval $[i+1, j]$.

For a codeword with circular span $[j, i]$ the elementary tail–biting trellis has profile $(q^{i+1}, 1^{j-i}, q^{n-j-1})$.This is abbreviated as $([q, 1, q], [j+1, i])$, as the state cardinality remains constant at q over the circular interval $[j+1, i]$. It is clear that the product construction will yield linear trellises where the state cardinality at any time index is always a power of q, where the code is over $GF(q)$.

We now describe an algorithm for the construction of minimal tail–biting trellises, for a subclass of cyclic codes. We establish the basis for the construction by a sequence of results. Firstly we recall, that the generator matrix formed by the vector representation of $g(x), xg(x), x^2 g(x), \ldots x^{k-1} g(x)$ gives us the minimal conventional trellis for the code. We aim to get a minimal tail-biting trellis having the same product space size as this one. Since each row has linear span of width $(n-k)$, and there are k rows, the product of the state cardinalities is $q^{k(n-k)}$. Since this is a minimal trellis, we cannot do better. Consequently, a lower bound for the maximum size of the state space at any time index is $q^{\lceil k(n-k)/n \rceil}$. This improves the total span lower bound on S_{max} for this class, as $n - k \geq d - 1$. We also note that any row with span (linear or circular) exceeding $n-k$ will increase the product state space size of the code. We can therefore restrict our attention to rows having span of width $n-k$. We prove the main result through

a sequence of lemmas. We assume that the codes under consideration are (n, k) cyclic codes.

Lemma 1. *Let $gcd(n, k) = 1$. Then the minimal tail-biting trellis for the cyclic code cannot be flat.*

Proof. Each row contributes a factor q^{n-k} to the product of the state cardinality and there are k rows. Thus the total product state cardinality is $q^{k(n-k)}$. For a flat trellis this must be distributed evenly among the n columns. Thus n must divide $k(n-k)$ which implies n must divide k^2. Since $gcd(n, k) = 1$, this is not possible, and hence a flat trellis cannot exist for this case.

Lemma 2. *For a minimal non-flat trellis for a cyclic code, $log(S_{max}) - log(S_{min})$*
≥ 1.

Proof. The proof follows from the fact that the state cardinality at each time index is always a power of q.

A trellis which has r jumps in its state cardinality profile is called an r-jump trellis. For a given trellis, if $log(S_{max}) - log(S_{min}) = \delta$, the trellis is called a δ-trellis. The tail-biting trellis for the Hamming code in Figure 6 is a 2-jump 1-trellis. We will be dealing with very restricted kinds of tail-biting trellises.These will have state cardinality profiles of the form $([q^l, q^{l+1}, q^l], [i, j])$ or $([q^{l+1}, q^l, q^{l+1}], [j, i])$. The first trellis is of type LHL(L for Low , H for High) and the second of type HLH. We will sometimes use the notation (LHL, i, j, δ) or (HLH, i, j, δ) to refer to 2-jump δ-trellises where the state cardinality remains constant at S_{max} over the linear interval $[i, j]$ or circular interval $[j, i]$ as the case may be. Note that 1-jump trellises do not exist and 0-jump trellises are flat.

We next present a technique to construct a minimal tail-biting trellis for a (n, k) cyclic code when $gcd(n, k) = 1$.

Lemma 3. *The trellis product of elementary trellises corresponding to the code polynomials $g(x), x^{(n-k)}g(x), x^{2(n-k)}g(x), \ldots, x^{(k-1)(n-k)}g(x)$ where all the products are modulo $x^n - 1$, yields a 2-jump 1-trellis with $log(S_{max}) = \lceil (k(n-k)/n) \rceil$.*

Proof. We construct the product trellis step-by-step, forming a trellis product by including one elementary trellis at each step in the order above. At each step the trellis is shown to be a 2-jump 1-trellis. We prove the following by induction.

Hypothesis The trellis generated by the first i codewords $1 \leq i \leq k$ is a 2-jump 1-trellis of type $(LHL, 1, (i)(n-k) \bmod n, 1)$, with $log(S_{max}) = \lceil (i(n-k)/n) \rceil$.

Basis i=1 The codeword corresponding to $g(x)$ has linear span $[0, n-k]$, and generates a $(LHL, 1, n-k, 1)$ trellis. Also $log(S_{max}) = 1$, proving the hypothesis for this case.

Induction Assume the hypothesis is true for the product of the first i or fewer trellises. Thus the trellis generated after i steps is a $(LHL, 1, (i)(n-k) \bmod n, 1)$ trellis, with $log(S_{max}) = \lceil (i(n-k)/n) \rceil$. The next codeword to be added is $x^{(i)(n-k)}g(x) \bmod (x^n - 1)$. If this has linear span, the trellis is of type

$(LHL, (i)(n-k)+1 \bmod n, (i+1)(n-k) \bmod n, 1)$. The product trellis is then of the type $(LHL, 1, (i+1)(n-k) \bmod n, 1)$ and there is no increase in S_{max}. If the next codeword has a circular span, then its trellis is of type $(HLH, (i)(n-k)+1 \bmod n, (i+1)(n-k) \bmod n, 1)$. (Here $i(n-k) \bmod n > (i+1)(n-k) \bmod n$.) Thus the jumps in the product trellis will be at 1 and $(i+1)(n-k)+1 \bmod n$. Also $log(S_{max})$ increases by 1 in this case, as the number of states from indices $i(n-k) \bmod n$ to $n-1$, and 0 to $(i+1)(n-k) \bmod n$ increases by a factor of q. We see that whenever the new elementary trellis to be added has circular span, the value of S_{max} increases by a factor of q. This happens $\lceil k(n-k)/n \rceil$ times. After adding k elementary trellises, the tail-biting trellis construction is complete. The trellis so constructed achieves the improved lower bound on S_{max} for cyclic codes.

While we have proved that the k codewords selected by the lemma do indeed give a trellis with a minimal value of S_{max} we need to show that they form a basis for the subspace defining the code. Thus we need to prove that they are linearly independent. For the next lemma we make the assumption that $gcd(q, n) = 1$. This implies that the generator and parity check polynomials do not have repeated roots. Most practical cyclic codes satisfy this property.

Lemma 4. *If $gcd(n, k) = 1$ and $gcd(q, n) = 1$ then the vectors corresponding to codewords $g(x), x^{(n-k)}g(x), x^{2(n-k)}g(x), \ldots, x^{(k-1)(n-k)}g(x)$ where all the products are modulo $x^n - 1$, are linearly independent.*

Proof. Assume that the vectors are linearly dependent.Then there exist scalars $a_0, a_1, \ldots, a_{k-1}$ in the field $GF(q)$ such that

$$(a_0 + a_1 x^{n-k} + \ldots + a_{k-1} x^{(k-1)(n-k)})(g(x)) \equiv 0 \; mod(x^n - 1) \qquad (1)$$

Let $A(x) = a_0 + a_1 x^{n-k} + \ldots + a_{k-1} x^{(k-1)(n-k)}$. From (1) and from the definition of $h(x)$, we conclude that $A(x)$ has as roots all the k roots of $h(x)$. Define $b(x) = a_0 + a_1 x + \ldots + a_{k-1} x^{k-1}$. If $\beta \in GF(q^m)$ is a root of $A(x)$, then β^{n-k} is a root of $b(x)$. Also if β_1, β_2 are distinct roots of $h(x)$, β_1^{n-k}, β_2^{n-k} are distinct roots of $b(x)$. Else, if $\beta_1 = \beta^{i_1}$ and $\beta_2 = \beta^{i_2}$ where β generates the cyclic group of order n, $\beta^{(i_1 - i_2)(n-k)} = 1$ implying that $i_1 - i_2 \equiv 0 (\bmod\, n)$, thereby giving a contradiction. Hence the vectors are linearly independent.

We finally present the main theorem.

Theorem 1. *For an (n, k) cyclic code over $GF(q)$ with $gcd(q, n) = 1, gcd(n, k) = 1$, there exists a choice of spans, such that the product of elementary trellises corresponding to the codewords $g(x), x^{(n-k)}(g(x)), x^{2(n-k)}(g(x)), \ldots, x^{(k-1)(n-k)}$ $(g(x))$ gives a Δ-minimal trellis for the code which is also Σ-minimal.*

Proof. The Δ-minimality follows from Lemmas 1, 2, 3 and 4. We next show that the Δ-minimal trellises constructed using the procedure above are also $\Sigma-$minimal. The trellis, say T_1, constructed by the above method has $log(S_{max}) - log(S_{min}) = 1$. Assume some other tail biting trellis(say T_2) which is π-minimal, also has $log(S_{max}) - log(S_{min}) = 1$. We show that such a trellis

has a state cardinality profile that is a permutation of T_1. Since a flat trellis does not exist, this trellis is $\Sigma - minimal$. Since $log(S_{max}) - log(S_{min}) = 1$ in T_1, it has a state cardinality of q^{m_1-1} at t_1 time indices and q^{m_1} at $n-t_1$ time indices for some m_1. Similarly T_2 has a state cardinality of q^{m_2-1} at t_2 time indices and q^{m_2} at $n-t_2$ time indices for some m_2,where $m_1, m_2 \geq 1$ and $t_1, t_2 \geq 1$. Without loss of generality assume $t_1 \geq t_2$. Since both of them are π-minimal,we have
$(m_1 - 1) * (t_1) + (m_1) * (n - t_1) = (m_2 - 1) * (t_2) + (m_2) * (n - t_2)$
$(m_1 * n) - t_1 = (m_2 * n) - t_2$
$n * (m_1 - m_2) = t_1 - t_2$
Since $t_1 - t_2 < n$, this is only possible if $t_1 - t_2 = 0$ implying that $m_1 = m_2$. This proves that for a given (n, k) cyclic code, with $gcd(n, k) = 1$, assuming π-minimality, the Σ-minimal trellis is unique up to permutations of the state cardinality profile and is the same as the Δ-minimal trellis.

Figure 6 is the minimal tail-biting trellis for the (7,4) cyclic Hamming code using our technique. We next turn our attention to cyclic codes for which $gcd(n, k) = t = gcd(n, n - k) > 1$. We use the property that if $gcd(n, k) = t$ then the smallest multiple of $n - k$ which is a multiple of n is $(n/t) \times (n - k)$.

Lemma 5. *Let $g(x)$ be the generator polynomial of an (n, k) cyclic code with $gcd(n, k) = t > 1$. Also assume that $k < n/t$. Then the codewords corresponding to the polynomials $g(x), x^{(n-k)}g(x), x^{2(n-k)} \ldots x^{(k-1)(n-k)}$ where all products are modulo $x^n - 1$, generate a minimal 2-jump 1-trellis for the code, provided the following condition is satisfied: if β^i and β^j are roots of $h(x)$ where β generates the cyclic group of order n, then $i - j \neq 0 (mod\ n/t)$.*

Proof. From Lemma 3, we know that the vectors generate a trellis which is a 2-jump 1-trellis that is Σ-minimal as well as Δ-minimal. To show that it is a trellis for the code, we need to show that the k vectors are linearly independent. Assume they are not. The proof proceeds along exactly the same lines as that of Lemma 4, with $A(x), b(x), \beta^i, \beta^j$ being defined as in that lemma. If β^i and β^j map to the same root of $b(x)$ then $(i - j)(n - k) = 0(\ \mathrm{mod}\ n)$. But the smallest value of $i - j$ for which this can happen is $i - j = n/t$. Since this is not possible, the k roots of $h(x)$ map into distinct roots of $b(x)$ giving a contradiction, as the degree of $b(x)$ is $k - 1$. Hence the vectors are linearly independent and generate a minimal trellis for the code.

4 Minimal Tail-Biting Trellises for Reed-Solomon Codes

We now present some results on mimimal trellises for Reed-Solomon codes. We recall that for such codes the code symbols and the roots of $g(x)$ are in the same field. We look at cases when $gcd(n, k) = t > 1$, as the case when t is 1 is covered by the results of the previous section. It is easy to prove that minimal flat trellises exist for several subclasses of Reed-Solomon codes.

The lemma below has a simple proof available in [4]. For the sake of brevity the proof is not reproduced here.

Lemma 6. *Let $g(x)$ be the generator polynomial for an (n,k) Reed-Solomon code. Then the codewords represented by polynomials $g(x), x^{i_1}g(x), x^{i_2}g(x), \ldots, x^{i_{k-1}}g(x)$ where $i_1, i_2, \ldots, i_{k-1}$ are distinct positive integers between 1 and n-1(inclusive),(and all products are modulo $x^n - 1$), are linearly independent.*

Lemma 7. *Let $t = gcd(n,k)$. Then the integers $i(n-k) \bmod n, i = 0, 1, \ldots n/t-1$ are all distinct, and the vectors corresponding to $g(x), x^{n-k}g(x), x^{2(n-k)}g(x), \ldots, x^{(n/t-1)(n-k)}g(x)$, where all the products are modulo x^n-1, generate a flat trellis.*

Proof. We can easily see that the integers are all distinct. The construction procedure of Lemma 3 will after $n/t-1$ products, generate a trellis with a "jump" up at index 1 and a "jump" down at index $n/t(n-k)+1 = 1$ modulo n. Thus after the $n/t - 1^{st}$ product, the trellis will be a 0-trellis and is hence flat.

Lemma 8. *Assume that $n/t = k$. Then the vectors $g(x), x^{n-k}g(x), x^{2(n-k)}g(x), \ldots, x^{(k-1)(n-k)}g(x)$ where all products are modulo $x^n - 1$, generate a minimal flat trellis for the code. If $n/t > k$, then the vectors generate a minimal 2-jump 1-trellis for the code.*

Proof. We can easily see that in the first case n divides k^2. By Lemmas 6 and 7 we see that the k vectors are distinct, linearly independent and also generate a minimal flat trellis for the code. The proof for the second case follows directly from Lemmas 3 and 6.

Lemma 9. *Let $t = gcd(n,k)$. Then if n/t divides k, the vectors in the union of the sets below:*
$\{g(x), x^{n-k}g(x) \ldots x^{(n/t-1)(n-k)}g(x)\}$
$\{xg(x), x^{n-k+1}g(x) \ldots x^{(n/t-1)(n-k)+1}g(x)\}$
$\{x^2g(x), x^{n-k+2}g(x) \ldots x^{(n/t-1)(n-k)+2}g(x)\}$
$\vdots$
$\{x^{(k/(n/t))-1}g(x), x^{n-k+(k/(n/t))-1}g(x) \ldots x^{(n/t-1)(n-k)+(k/(n/t))-1}g(x)\}$
generate a flat trellis for the code.

Proof. We outline the proof here. Firstly we see that n divides k^2. Also, it is easy to see that the vectors are all distinct. We note from Lemma 7, that the vectors in each set produce a flat trellis of width $q^{((n-k)\times(n/t))/n}$. (Cyclic shifts of a generator matrix that produces a flat trellis also produce flat trellises). The product trellis is just a product of flat trellises, and is hence also flat. To see that it is minimal, we see that the width is $q^{1/n((n/t)\times(n-k)\times k/(n/t))} = q^{k(n-k)/n}$. Thus the trellis is minimal.

The only case left is that when $k > n/t$ but n/t does not divide k. It is easy to see that in this case we will get $\lceil (k/(n/t)) \rceil$ sets, such that the last set contains less than n/t vectors, and generates a 2-jump 1-trellis, while the others all generate flat trellises. The result follows from these two observations. Thus we see that all cases for Reed-Solomon codes are covered. We consolidate all the results of this section into one theorem.

Theorem 2. *Reed-Solomon (n,k) codes have minimal flat trellises if and only if n divides k^2. If n does not divide k^2 the minimal trellises are 2-jump 1-trellises.*

Proof. The necessity of the condition for flat trellises is shown in Lemma 1. Let $gcd(n,k) = t$, and assume n divides k^2. We can write $n_1 \times n = k^2 = k_1 \times t \times k$, for integers n_1 and k_1. Clearly $n_1/k_1 = l$, is an integer, as $gcd(k_1, n) = 1$. Therefore $l \times n/t = k$ which, together with lemmas 8 and 9 show the sufficiency of the condition for a flat trellis. The case n does not divide k^2 is covered by lemma 3, the second half of lemma 8 and the previous paragraph.

References

1. L.R.Bahl, J.Cocke, F.Jelinek, and J. Raviv, Optimal decoding of linear codes for minimizing symbol error rate, *IEEE Trans. Inform. Theory* **20**(2), March 1974, pp 284-287.
2. A.R.Calderbank, G.David Forney,Jr., and Alexander Vardy, Minimal Tail-Biting Trellises: The Golay Code and More, *IEEE Trans. Inform. Theory* **45**(5) July 1999, pp 1435-1455.
3. G.D. Forney, Jr. and M.D. Trott, The dynamics of group codes:State spaces, trellis diagrams and canonical encoders, *IEEE Trans. Inform. Theory* **39**(5) Sept 1993, pp 1491-1513.
4. Harmeet Singh, On Tail-Biting Trellises for Linear Block Codes, M.E. Thesis, Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore, 2001.
5. Ralf Kotter and Vardy, A.,Construction of Minimal Tail-Biting Trellises, in *Proceedings IEEE Information Theory Workshop* (Killarney, Ireland, June 1998), 72-74.
6. F.R.Kschischang and V.Sorokine, On the trellis structure of block codes, *IEEE Trans. Inform. Theory* **41**(6), Nov 1995, pp 1924-1937.
7. F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error Correcting Codes*, North-Holland, Amsterdam, 1981.
8. J.L.Massey, Foundations and methods of channel encoding, in *Proc. Int. Conf. on Information Theory and Systems* **65**(Berlin, Germany) Sept 1978.
9. R.J.McEliece, On the BCJR trellis for linear block codes, *IEEE Trans. Inform. Theory* **42**, November 1996, pp 1072-1092.
10. D.J. Muder, Minimal trellises for block codes, *IEEE Trans. Inform. Theory* **34**(5), Sept 1988, pp 1049-1053.
11. Priti Shankar, P.N.A. Kumar, K.Sasidharan and B.S.Rajan, ML decoding of block codes on their tail-biting trellises, to appear in *Proceedings of the 2001 IEEE International Symposium on Information Theory.*
12. Yaron Shany and Yair Be'ery, Linear Tail-Biting Trellises, the Square-Root Bound, and Applications for Reed-Muller Codes, *IEEE Trans. Inform. Theory* **46** (4), July 2000, pp 1514-1523.
13. A.Vardy, Trellis structure of codes, in *Handbook of Coding Theory*,V.S. Pless and W.C. Huffman, Eds., Elsevier Science, 1998.
14. N.Wiberg, H.-A. Loeliger and R.Kotter, Codes and iterative decoding on general graphs, *Eoro. Trans. Telecommun.*,**6**, Sept 1995, pp 513-526.
15. J.K. Wolf, Efficient maximum-likelihood decoding of linear block codes using a trellis, *IEEE Trans. Inform. Theory* **24**, pp 76-80.

Effective Lossy Queue Languages

Parosh Aziz Abdulla[1], Luc Boasson[2], and Ahmed Bouajjani[2]

[1] DoCS, Uppsala University, P.O. Box 325, S-75105 Uppsala, Sweden.
parosh@docs.uu.se

[2] Liafa, Univ. of Paris 7, Case 7014, 2 place Jussieu, F-75251 Paris Cedex 05, France.
{lub,abou}@liafa.jussieu.fr

Abstract. Although the set of reachable states of a lossy channel system (LCS) is regular, it is well-known that this set cannot be constructed effectively. In this paper, we characterize significant classes of LCS for which the set of reachable states can be computed. Furthermore, we show that, for slight generatlizations of these classes, computability can no longer be achieved.
To carry out our study, we define *rewriting systems* which capture the behaviour of LCS, in the sense that (i) they have a FIFO-like semantics and (ii) their languages are downward closed with respect to the substring relation. The main result of the paper shows that, for context-free rewriting systems, the corresponding language can be computed. An interesting consequence of our results is that we get a characterization of classes of meta-transitions whose post-images can be effectively constructed. These meta-transitions consist of sets of nested loops in the control graph of the system, in contrast to previous works on meta-transitions in which only single loops are considered.
Essentially the same proof technique we use to show the result mentioned above allows also to establish a result in the theory of $0L$-systems, i.e., context-free parallel rewriting systems. We prove that the downward closure of the language generated by any $0L$-system is effectively regular.

1 Introduction

We consider the problem of model checking of *lossy channel systems (LCS)* which consist of finite-state processes communicating over FIFO-buffers. The buffers are unbounded and *lossy* in the sense that they can lose messages. Such systems can be used to model the behaviour of communication protocols such as link protocols [AJ96] and bounded retransmission protocols [GvdP93,AAB99] which are designed to operate correctly even in the case where the underlying communication medium cannot provide reliable communication. In [AJ96], an algorithm for checking safety properties for LCS is described. The algorithm performs a fixed point iteration where each iteration computes the pre-image of a set of configurations with respect to a single transition of the system. This corresponds to a backward reachability analysis algorithm which constructs a characterization (as a regular set) of the set of configurations from which a set of *final configurations* is reachable.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 639–651, 2001.

Often, it is also important to be able to perform a forward reachability analysis, and characterize the set of configurations which are reachable from the initial configuration of the system. For instance, several efficient verification methods, such as on-the-fly algorithms [Hol91,CVWY90], are based on forward search of the state space. Also, using forward analysis, we can often automatically generate finite-state abstractions of the system. The abstract model may then be used to check several classes of properties which cannot be analyzed directly in the original model. It is well known that the set of reachable configurations in an LCS can be characterized as a regular set [CFI96,AB.I98]. On the other hand, using undecidability results reported in [May00], it can be shown that this characterization cannot be effectively constructed. Therefore, it is interesting to characterize the class of LCS for which the characterization is effective.

To achieve that, we model the behaviour of an LCS as a rewriting system. To reflect the behaviour of the LCS, we provide a FIFO-like semantics for the rewriting system, i.e., when a rule is applied to a string, the left hand side is removed from the head of the string, while the right hand side is appended to the end of the string. Furthermore, corresponding to lossiness, we require that the language generated is downward closed with respect to the substring relation. Intuitively, the symbols in the left hand side of a rule correspond to the messages received from the channels, while the symbols in the right hand side correspond to messages sent to the channels. This implies that characterizing the class of rewriting systems with effectively computable languages gives also a characterization of the class of LCS with computable reachability sets.

In this paper, we study the limits of the computability of the reachability set, and characterize significant classes of rewriting systems which have effectively computable languages. The main result of the paper is showing that all context-free rewriting systems have effectively computable languages. Furthermore, we show that slight generalizations of the context-free class lead to uncomputability of the language. For instance, it is sufficient to allow two symbols in the left hand sides of the rewriting rules in order to lose computability. Moreover, if we consider rewriting systems on vectors of words (representing FIFO-channel systems with several queues), then computability is lost starting from dimension 2. Nevertheless, we show that for particular n-dim context-free rewriting systems called *rings*, the generated language is effectively computable. The class of rings corresponds to n-dim systems where each rewriting rule consists in receiving a message from some channel of index say i, and sending a string (sequence of messages) to the channel of index $(i + 1)$ mod n. These rules correspond to actions of communicating processes connected according to a ring topology, where each component has an input and an output buffer.

A significant application of our results is to characterize classes of *meta-transitions* which can be used in forward reachability analysis of LCS. A *meta-transition* is an *arbitrarily long sequence* of transitions, which often corresponds to repeated executions of loops in the syntactic code of the program. The idea is to speed up the iterations corresponding to forward reachability analysis by computing the post-image of a *meta-transition* rather than a single transition.

In almost all practical cases, applying meta-transitions is necessary to achieve termination of the iteration procedure. Obviously, the key issue is to decide which meta-transitions to apply during each iteration step. A necessary criterion is that a meta-transition should allow computability of post-images, i.e., the post-image of each constraint with respect to the meta-transition should be computable. Since the set of transitions in a meta-transition can themselves be viewed as an LCS, our result also gives a characterization of meta-transitions which allow computing of post-images. Observe that the language generated describes the application of an arbitrary sequence of rules, which means that we are able to describe the effect of arbitrary sequences of control loops. In this way we extend significantly the class of LCS on which accelerations can be applied, since earlier works [BG96,BGWW97,BH97,ABJ98] only consider single loops.

It turns out that our proof can also be adapted to establish a result in the language theoretical framework of $0L$-systems [RS80]. These systems consist of context-free sets of rules with a parallel rewriting relation. In fact, there is a tight relation between FIFO rewriting and parallel rewriting (which is actually used in our proof). We show that the set of subwords of the language generated by any $0L$-system is an effective regular set.

Related Work: Several works have studied computing post-images of meta-transitions in the context of communicating finite-state processes [BG96, BGWW97] [BH97,ABJ98]. In contrast to this paper, all these works consider meta-transitions which correspond to *single loops*. This excludes several common structures which occur in protocols such as *nested loops*. An example of a nested loop is two transitions sending two different messages to the channels. For instance, in [ABJ98], we show how to compute post-images of single loops for lossy channel systems. In [BGWW97], a complete characterization is given of the class of simple loops which preserve regularity in the context of systems with *perfect channels*. In [BH97] a constraint system is presented and shown to be closed under the execution of any single loop in a system with perfect channels.

Context-free rewriting systems have been used for description and analysis for several classes of infinite-state systems such as BPA and pushdown processes (e.g., [BBK93,CHS92,GH94,Sti96]). All these works deal with the classical (stack-like) semantics, and hence cannot be applied to analyze systems with FIFO buffers. [Bur98] considers rewriting systems with *perfect* FIFO behaviour. It shows that all interesting verification problems (including reachability) are undecidable for these systems.

$0L$-systems have been studied as an alternative of the classical sequential rewriting systems, following the main lines suggested by the theory of context-free grammars. It has been shown that the properties of the two kinds of formalisms are quite different. In particular, the expressive power of the $0L$-systems is not comparable with the usual Chomsky hierarchy. Our result concerning $0L$-system corresponds to the result established in [Cou91] for the context-free languages.

Outline: In Section 2 we define some basic notions and introduce FIFO rewriting systems (cyclic rewriting systems) and $0L$-systems. In Section 3 we prove

our main result concerning the effective regularity of the language of downward closed context-free systems. We give also in this section our result on the effectiveness of the downward closure of $0L$ languages. In Section 4 we give unconstructibility results for downward closed rewriting systems with unrestricted (non context-free) rules, as well as for 2-dim context-free systems. Finally, in Section 5, we address the effective regularity of the n-dim context-free rings languages. For lack of space, some proofs are omitted in this version of the paper.

2 Preliminaries

In this section, we give some basic definitions and introduce the FIFO rewriting systems which we call *cyclic* rewriting systems.

2.1 Words and Languages

Let Σ be a finite alphabet. We denote by Σ^* the set of *words* over Σ, i.e., sequences (or strings) of symbols in Σ. We denote by ϵ the empty word.

We denote by $\preceq \subseteq \Sigma^* \times \Sigma^*$ the subword relation, i.e., $a_1 \dots a_n \preceq b_1 \dots b_m$ if there exists $i_1, \dots, i_n \in \{1, \dots, m\}$ such that $i_1 < \dots < i_n$ and $\forall j \in \{1, \dots, n\}$. $a_j = b_{i_j}$. We consider the product generalization of this relation to vectors of words.

An n-dim language over Σ, for $n \geq 1$, is any subset of $(\Sigma^*)^n$. 1-dim languages are called languages as usual. An n-dim language L is *downward closed* if $\forall \boldsymbol{u}, \boldsymbol{v} \in (\Sigma^*)^n$, if $\boldsymbol{v} \in L$ and $\boldsymbol{u} \preceq \boldsymbol{v}$, then $\boldsymbol{u} \in L$. Let $L{\downarrow}$ denote the *downward closure* of L, i.e., the smallest downward closed set which includes L. Notice that a set L is downward closed if and only if $L = L{\downarrow}$.

2.2 Simple Regular Expressions

Let us call *atomic expression* any expression e of the form $(a + \epsilon)$ where $a \in \Sigma$, or of the form A^* where $A \subseteq \Sigma$. A *product* p is either the empty word ϵ, or a finite sequence of the form $e_1 \cdots e_m$ of atomic expressions. A *simple regular expression* (SRE) is either $\emptyset$, or a finite union $p_1 + \cdots + p_n$ where each p_i is a product.

Theorem 1 ([ABJ98]). *SRE definable sets are precisely downward closed sets. Moreover, for every effective regular language L, $L{\downarrow}$ is an effective SRE language.*

Proposition 1 ([Cou91]). *For every context-free language L (effectively described by a context-free grammar), the set $L{\downarrow}$ is an effective regular set.*

2.3 Cyclic Rewriting Systems

For $n \geq 1$, an n-dim *rewriting rule* r over the alphabet Σ is a pair $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ where $\boldsymbol{x}, \boldsymbol{y} \in (\Sigma^*)^n$. We denote such a rule by $r : \boldsymbol{x} \mapsto \boldsymbol{y}$. The *left hand side* (resp. *right hand side*) of r, denoted by $lhs(r)$ (resp. $rhs(r)$), is the vector $\boldsymbol{x}$ (resp. $\boldsymbol{y}$).

An n-dim rewriting rule r is *context-free* if $lhs(r) \in (\Sigma \cup \{\epsilon\})^n$. A set of rules R is context-free if all its rules are context-free.

An n-dim *cyclic rewriting system* is a pair (R, R_c) where R is a finite set of n-dim rewriting rules, and $R_c : (\Sigma^*)^n \to 2^{(\Sigma^*)^n}$ is the *cyclic rewriting mapping* over R defined as follows: for every $\boldsymbol{u} = (u_1, \ldots, u_n), \boldsymbol{v} = (v_1, \ldots, v_n) \in (\Sigma^*)^n$, we have $\boldsymbol{v} \in R_c(\boldsymbol{u})$ if and only if there exists a rule $r : (x_1, \ldots, x_n) \mapsto (y_1, \ldots, y_n) \in R$ such that $\forall i \in \{1, \ldots, n\}.\ \exists w_i \in \Sigma^*.\ u_i = x_i w_i$ and $v_i = w_i y_i$.

A *weak cyclic rewriting system* is a pair (R, R_{wc}) where R is a finite set of n-dim rules and R_{wc} is the *weak cyclic rewriting mapping* over R defined as follows: for every $\boldsymbol{u}, \boldsymbol{v} \in (\Sigma^*)^n$, we have $\boldsymbol{v} \in R_{wc}(\boldsymbol{u})$ if and only if there exist $\boldsymbol{u}', \boldsymbol{v}' \in (\Sigma^*)^n$ such that $\boldsymbol{u}' \preceq \boldsymbol{u}$, $\boldsymbol{v}' \in R(\boldsymbol{u}')$, and $\boldsymbol{v} \preceq \boldsymbol{v}'$.

The definition of the mappings R_c and R_{wc} are extended straightforwardly to sets of vectors of words. We denote by R_c^* and R_{wc}^* the reflexive-transitive closure of R_c and R_{wc} respectively. Given a language L, $R_c^*(L)$ (resp. $R_{wc}^*(L)$ is called its *cyclic closure* (resp. *weak cyclic closure*) by R.

Cyclic rewriting systems can be used to model processes communicating through unbounded FIFO queues (channels). Intuitively, a rule $r : (x_1, \ldots, x_n) \mapsto (y_1, \ldots, y_n)$ corresponds to an action which, for a queue of index i, receives the sequence x_i from the queue, and then sends the sequence y_i to the queue. Weak cyclic rewriting systems model processes communicating through unbounded *lossy* channels (which may lose messages at any time).

Cyclic rewriting systems correspond to Post tag systems. It is well known that they are Turing machine powerful, and hence, the set $R_c^*(L)$ is in general not constructible, for any language L. On the other hand, we know by Theorem 1 that for every n-dim language L and every set of rewriting rules R, the set $R_{wc}^*(L)$ is SRE definable. However, this set is not necessarily effectively constructible.

2.4 Substitutions and 0*L*-Systems

A *substitution* is a mapping $\phi : \Sigma \to 2^{\Sigma^*}$ which associates with each symbol $a \in \Sigma$ a language over Σ (which may be empty).

A substitution is extended to words by taking $\phi(a_1 \cdots a_n) = \phi(a_1) \cdots \phi(a_n)$ and $\phi(\epsilon) = \epsilon$, and then generalized straightforwardly to languages. We denote by ϕ^i the substitution obtained by i compositions of ϕ. Then, let $\phi^{\geq n}(a)$ denote the set $\bigcup_{i \geq n} \phi^i(a)$, for any $n \geq 0$ and $a \in \Sigma$. We denote by ϕ^* (resp. ϕ^+) the reflexive-transitive (resp. transitive) closure of ϕ, i.e., $\phi^{\geq 0}(a)$ (resp. $\phi^{\geq 1}(a)$).

A substitution ϕ is *finite* if, for every $a \in \Sigma$, $\phi(a)$ is a finite language. A substitution is *downward closed* if it associates with each symbol a downward closed language, and it is *complete* if, for every $a \in \Sigma$, $\phi(a)$ is nonempty. A *weak substitution* is a downward closed complete substitution.

Given a set of 1-dim context-free rewriting rules R, we define a substitution φ_R such that, for every $a \in \Sigma$, $\varphi_R(a) = \{rhs(r) \ :\ r \in R \text{ and } lhs(r) = a\}$. We define also a substitution ϕ_R such that, for every $a \in \Sigma$, $\phi_R(a) = \{\epsilon\} \cup \varphi_R(a)\downarrow$. Notice that, since R is finite, both φ_R and ϕ_R are finite substitutions. Moreover, ϕ_R is a weak substitution whereas φ_R is in general not necessarily complete and

not downward closed. Since φ_R is not complete, $\varphi_R(a)\downarrow$ is in general different from $\phi_R(a)$ (since $\emptyset\downarrow = \emptyset$).

A $0L$-*system* (resp. *weak* $0L$-*system*) is a pair (R, φ_R) (resp. (R, ϕ_R)) where R is a finite set of context-free rewriting rules.

The following lemma shows the link between cyclic context-free rewriting and substitution (parallel rewriting).

Lemma 1. *For every finite set of 1-dim context-free rules R, and every language L, if $\phi_R^*(L)$ is an effective SRE set, then $R_{wc}^*(L)$ is also an effective SRE set.*

3 Constructibility for Context-Free Systems

We give in this section our main result (Theorem 2) and present its proof. This result says that the language generated by a weak cyclic context-free rewriting system is an effective regular (actually SRE) set.

An adaptation of the proof for this theorem can be done to establish a result concerning the languages generated by $0L$-systems (Theorem 3). The result says that the set of subwords of such a language is effectively regular.

Theorem 2. *For every context-free language L (effectively described by a context-free grammar), and for every finite set of (1-dim) context-free rewriting rules R, the set $R_{wc}^*(L)$ is effectively SRE representable.*

Proof. Clearly, $R_{wc}^*(L) = R_{wc}^*(L\downarrow)$ for every language L. Hence, by Proposition 1 and Theorem 1, it is sufficient to prove that $R_{wc}^*(L)$ is an effective SRE set for every SRE language L. Moreover, by Lemma 1, this can be done by proving that $\phi_R^*(L)$ is an effective SRE set. For that, we need some definitions.

Given a finite weak substitution ϕ (effectively given), we define $\Sigma_\phi(a) = \phi^*(a) \cap \Sigma$. Intuitively, $\Sigma_\phi(a)$ is the set of accessible symbols from a by iterated substitutions. Notice that the set $\Sigma_\phi(a)$ contains the symbol a itself. Notice also that this set can be computed since the alphabet Σ is finite.

We partition the set of symbols Σ according to the following equivalence relation: two symbols a and b are equivalent if $\Sigma_\phi(a) = \Sigma_\phi(b)$. The equivalence class of a symbol a is denoted $[a]$.

The accessibility relation induces an ordering between equivalences classes: $[a] \leq [b]$ if and only if $\Sigma_\phi(a) \subseteq \Sigma_\phi(b)$. Let $level(a)$ be the level of $[a]$ according to this ordering: minimal elements (equivalence classes) have level 0, and level $k+1$ corresponds to minimal elements of the set of classes without the elements of level less than k.

Given a finite weak substitution ϕ, a symbol a is *recursive* if $a \in \phi^+(a)$. It is *expansive* if $aa \in \phi^+(a)$. A symbol is *linear* if it is recursive and not expansive. Let us denote by Rec_ϕ (resp. Exp_ϕ, Lin_ϕ, $NRec_\phi$) the set of recursive (resp. expansive, linear, nonrecursive) symbols according to ϕ. Deciding whether a symbol is recursive, linear, or expansive is straightforward. Notice that for every $a, b \in \Sigma$ such that $[a] = [b]$, we have $a \in Rec_\phi$ (resp. Exp_ϕ, Lin_ϕ, $NRec_\phi$) iff $b \in Rec_\phi$ (resp. Exp_ϕ, Lin_ϕ, $NRec_\phi$).

A substitution ϕ is *normal* if, for every $a \in Exp_\phi$ (resp. Lin_ϕ), the set $\phi(a)$ contains aa (resp. a).

Let R be a finite set of 1-dim context-free rewriting rules. We suppose hereafter that R does not contain rules of the form $\epsilon \mapsto y$. The consideration of this kind of rules is not difficult.

For every $a \in Exp_{\phi_R}$ (resp. Lin_{ϕ_R}), let $\pi(a)$ be the smallest strictly positive integer such that aa (resp. a) is in $\phi_R^{\pi(a)}(a)$. Then, let $\pi = \text{lcm}\{\pi(a) : a \in Rec_{\phi_R}\}$, and let $\psi_R = \phi_R^\pi$. It is easy to see that ψ_R is a normal finite weak substitution. Since for every symbol $a \in \Sigma$, we have $\phi_R^*(L) = \bigcup_{i=1}^{\pi-1} \phi_R^i(\psi_R^*(L))$, we deduce that:

Lemma 2. *For every finite set of 1-dim context-free rewriting rules R, and every language L, if the set $\psi_R^*(L)$ is effective, then the set $\phi_R^*(L)$ is effective.*

By Lemma 2, and since ψ_R is a normal finite weak substitution, in order to compute the image by ϕ_R^* of a language L, it is sufficient to know how to compute the image of L for every given normal finite weak substitution. The remainder of the proof consists in showing that this is possible. For the sake of simplicity, we show this fact for L reduced to a symbol. The generalization to any regular language is not difficult. Therefore, we prove the following key proposition.

Proposition 2. *For every normal finite weak substitution ϕ, and every symbol $a \in \Sigma$, the set $\phi^*(a)$ is effectively SRE representable.*

Proof. To present the proof, we need to introduce some notions and to establish several lemmas. Let ϕ be a substitution. Then, for any positive integer B, we say that a symbol a is B-regular if $\phi^{B+n}(a) \subseteq \phi^{B+n+1}(a)$ for every $n \geq 0$. This means that, after the B first applications of ϕ, the iteration of ϕ yields a non-decreasing sequence of sets (i.e., each application of ϕ augments or at least preserves the given set).

Remark 1. By definition, if a symbol is B-regular, then it is necessarily B'-regular for every $B' \geq B$.

Let us consider some simple facts. The first one is that, since $a \in \phi(a)$, all recursive symbols (hence, all linear and expansive symbols) are 0-regular.

Lemma 3. *Let ϕ be normal finite weak substitution. Then, for every $a \in Rec_\phi$, a is 0-regular.*

Next lemma says that $\phi^*(a)$ is effective when $a \in Exp_\phi$.

Lemma 4. *Let ϕ be a normal finite weak substitution. Then, for every $a \in Exp_\phi$, $\phi^*(a) = (\Sigma_\phi(a))^*$.*

Now, we give the main lemma which allows to prove Proposition 2. The proof of this lemma itself involves several lemmas (Lemmas 6, 7, 8, 9, 10, 11, and 12) which allow to establish that $\phi^*(a)$ is effective when $a \in Lin_\phi$.

Lemma 5. *Let ϕ be a normal finite weak substitution. Then, for every $a \in \Sigma$, there exists a $B \geq 0$ such that a is B-regular and $\phi^*(a)$ is effectively regular.*

Proof. The proof is by induction on the level of the symbols.

Basis: Let $a \in \Sigma$ such that $level(a) = 0$. If a is nonrecursive, then $\phi(a) = \{\epsilon\}$, and hence, a is 1-regular, and $\phi^*(a) = \{\epsilon\}$ is effectively regular. If a is expansive, then by Lemma 3 and Lemma 4, we know that a is 0-regular and $\phi^*(a)$ is effectively regular. If a is linear, then by Lemma 3 a is 0-regular. Moreover, since $level(a) = 0$, we have $\phi(a) = \Sigma_\phi(a)$, and hence $\phi^*(a) = [a]$ is effectively regular.

Inductive step: Let us now consider a symbol a such that $level(a) = k + 1$ for some $k \geq 0$, and assume, by induction hypothesis and Remark 1, that there exists $B \geq 0$ such that every symbol b of level at most k is B-regular, and $\phi^*(b)$ is effectively regular. We show that the symbol a is $(B + 1)$-regular and that $\phi^*(a)$ is effectively regular.

Case 1: a is nonrecursive. Then, for every $b \in \Sigma_\phi(a)$, $level(b) < level(a)$. By induction hypothesis, for every $b \in \Sigma_\phi(a)$, b is B-regular and $\phi^*(b)$ is effective. It easy to see that a is $(B + 1)$-regular and $\phi^*(a)$ is effectively regular.

Case 2: a is expansive. By Lemma 3, we know that a is 0-regular, and hence, a is $(B + 1)$-regular (for any $B \geq 0$). Moreover, by Lemma 4, we know that $\phi^*(a)$ is effectively regular.

Case 3: a is linear. We know by Lemma 3 that a is 0-regular, and hence that it is $(B + 1)$-regular. It remains to show that $\phi^*(a)$ is effectively regular.

Let us consider the context-free grammar $G_a = (N_a, T_a, S_a, \rho_a)$ such that:

- the set of nonterminal symbols is $N_a = [a]$,
- the set of terminal symbols is $T_a = \Sigma_\phi(a) \setminus [a]$,
- the start symbol is $S_a = a$,
- the set of rules is $\rho_a = \{b \to w \ : \ b \in [a] \text{ and } w \in \phi(b)\}$.

Notice that, since a is linear, the rules of G_a are either *nonterminal rules* of the form $b \to ub'v$, where $b' \in [a]$ and $u, v \in T_a^*$, or *terminal rules* of the form $b \to w$ where $w \in T_a^*$. Since ϕ is normal, we have by definition rules of the form $b \to b$ for any $b \in [a]$.

We index the nonterminal rule from 1 to n_1 and the terminal rules from 1 to n_2, where n_1 (resp. n_2) is the number of nonterminal (resp. terminal) rules in G_a. Then, we derive from G_a another context-free grammar $\widehat{G_a} = (N_a, \widehat{T_a}, S_a, \widehat{\rho_a})$ where:

- $\widehat{T_a} = \{\langle u\rangle_i, \langle v\rangle_i \ : \ \exists r = b \to ub'v \in \rho_a.\ index(r) = i\} \cup \{\langle\langle w\rangle\rangle_i \ : \ \exists r = b \to w \in \rho_a.\ w \in T_a^* \text{ and } index(r) = i\}$.
- $\widehat{\rho_a} = \{b \to \langle u\rangle_i b' \langle v\rangle_i \ : \ \exists r = b \to ub'v \in \rho.\ index(r) = i\} \cup \{b \to \langle\langle w\rangle\rangle_i \ : \ \exists r = b \to w \in \rho.\ w \in T_a^* \text{ and } index(r) = i\}$.

The following lemmas show the links between derivation in the grammar $\widehat{G_a}$ and the iterations of the substitution ϕ.

Lemma 6. *If there is a derivation in $\widehat{G_a}$ of length ℓ from a to some nonterminal word, then this word is of the form $\langle u_\ell\rangle_{i_\ell}\cdots\langle u_2\rangle_{i_2}\langle u_1\rangle_{i_1} b\langle v_1\rangle_{i_1}\langle v_2\rangle_{i_2}\cdots\langle v_\ell\rangle_{i_\ell}$, and*

$$\phi^{\ell-1}(u_\ell)\cdots\phi(u_2)u_1bv_1\phi(v_2)\cdots\phi^{\ell-1}(v_\ell)\subseteq\phi^\ell(a).$$

Lemma 7. *For every word $ubv\in\phi^\ell(a)$ where $b\in[a]$, there exists a derivation in $\widehat{G_a}$ of length ℓ from a to some nonterminal word of the form*

$$\langle u_\ell\rangle_{i_\ell}\cdots\langle u_2\rangle_{i_2}\langle u_1\rangle_{i_1} b\langle v_1\rangle_{i_1}\langle v_2\rangle_{i_2}\cdots\langle v_\ell\rangle_{i_\ell}$$

such that $u\in\phi^{\ell-1}(u_\ell)\cdots\phi(u_2)u_1$, and $v\in v_1\phi(v_2)\cdots\phi^{\ell-1}(v_\ell)$.

Lemma 8. *If there is a derivation in $\widehat{G_a}$ of length ℓ from a to some terminal word, then this word is of the form $\langle u_\ell\rangle_{i_\ell}\cdots\langle u_2\rangle_{i_2}\langle u_1\rangle_{i_1}\langle\langle w\rangle\rangle_i\langle v_1\rangle_{i_1}\langle v_2\rangle_{i_2}\cdots\langle v_\ell\rangle_{i_\ell}$, and*

$$\phi^\ell(u_\ell)\cdots\phi^2(u_2)\phi(u_1)w\phi(v_1)\phi^2(v_2)\cdots\phi^\ell(v_\ell)\subseteq\phi^\ell(a).$$

The proofs of the three lemmas above are by straightforward inductions.

Lemma 9. *For every terminal word σ in $\phi^\ell(a)$ (i.e., $\sigma\in T_a^*\cap\phi^\ell(a)$), there exists an integer $m\le\ell$ and a derivation in $\widehat{G_a}$ of length m from a to some terminal word $\mu=\langle u_m\rangle_{i_m}\cdots\langle u_2\rangle_{i_2}\langle u_1\rangle_{i_1}\langle\langle w\rangle\rangle_i\langle v_1\rangle_{i_1}\langle v_2\rangle_{i_2}\cdots\langle v_m\rangle_{i_m}$ such that*

$$\sigma\in\phi^{\ell-m}[\phi^m(u_m)\cdots\phi^2(u_2)\phi(u_1)w\phi(v_1)\phi^2(v_2)\cdots\phi^m(v_m)].$$

Proof. The fact that $\sigma\in\phi^\ell(a)$ implies that there are words $\sigma_0,\sigma_1,\ldots,\sigma_\ell$ such that $\sigma_0=a$, $\sigma=\sigma_\ell$, and $\sigma_{i+1}\in\phi(\sigma_i)$ for every $i\in\{0,\ldots,\ell-1\}$. Let σ_i be the last word in this sequence which contains a symbol $b\in[a]$. This means that after i applications of ϕ, we get a terminal word σ_{i+1}. Let $m=i+1$ and $\mu=\sigma_m$. By Lemma 7, we know that there exists a derivation in $\widehat{G_a}$ of length $i=m-1$ from a to some nonterminal word $\mu'=\langle u_\ell\rangle_{i_\ell}\cdots\langle u_2\rangle_{i_2}\langle u_1\rangle_{i_1}b\langle v_1\rangle_{i_1}\langle v_2\rangle_{i_2}\cdots\langle v_\ell\rangle_{i_\ell}$ such that $\mu\in\phi^{m-1}(u_m)\cdots\phi(u_2)u_1bv_1\phi(v_2)\cdots\phi^{m-1}(v_m)$. The result follows from the fact that μ can be derived in $\widehat{G_a}$ from μ' using a terminal rule, and from the fact that $\sigma\in\phi^{\ell-m}(\mu)$. □

Now, the following lemmas show the key property which allows to compute $\phi^*(a)$ by iterating the application of ϕ only a finite number of times, thanks to B-regularity.

Lemma 10. *If there is a derivation in $\widehat{G_a}$ of length ℓ from a to some nonterminal word $\langle u_\ell\rangle_{i_\ell}\cdots\langle u_2\rangle_{i_2}\langle u_1\rangle_{i_1}b\langle v_1\rangle_{i_1}\langle v_2\rangle_{i_2}\cdots\langle v_\ell\rangle_{i_\ell}$, then*

$$\phi^{\ge B}(u_\ell)\cdots\phi^{\ge B}(u_{B+1})\phi^{B-1}(u_B)\cdots\phi(u_2)u_1b\\ v_1\phi(v_2)\cdots\phi^{B-1}(v_B)\phi^{\ge B}(v_{B+1})\cdots\phi^{\ge B}(v_\ell)\subseteq\phi^*(a)$$

Proof. Suppose that there is a derivation δ in $\widehat{G_a}$ from a to the nonterminal word $\alpha = \langle u_\ell\rangle_{i_\ell}\cdots\langle u_2\rangle_{i_2}\langle u_1\rangle_{i_1} b \langle v_1\rangle_{i_1}\langle v_2\rangle_{i_2}\cdots\langle v_\ell\rangle_{i_\ell}$, and consider a word σ in

$$\phi^{\geq B}(u_\ell)\cdots\phi^{\geq B}(u_{B+1})\phi^{B-1}(u_B)\cdots\phi(u_2)u_1 b \\ v_1\phi(v_2)\cdots\phi^{B-1}(v_B)\phi^{\geq B}(v_{B+1})\cdots\phi^{\geq B}(v_\ell). \quad (1)$$

Hence, σ can be written $\mu_\ell\cdots\mu_{B+1}\mu_B\cdots\mu_2\mu_1 b\nu_1\nu_2\cdots\nu_B\nu_{B+1}\cdots\nu_\ell$ where $\mu_i \in \phi^{i-1}(u_i)$ and $\nu_i \in \phi^{i-1}(v_i)$ for every $i \in \{1,\ldots,B\}$, and for every $i \in \{B+1,\ldots,\ell\}$, there is $m_i, n_i \geq B$ such that $\mu_i \in \phi^{m_i}(u_i)$ and $\nu_i \in \phi^{n_i}(v_i)$. Let $\kappa = \max\{n_i, m_i \ : \ B+1 \leq i \leq \ell\}$. Then, using the property of B-regularity (see Remark 1), we deduce from (1) that σ is in the set:

$$\phi^{\kappa+\ell-(B+1)}(u_\ell)\cdots\phi^{\kappa}(u_{B+1})\phi^{B-1}(u_B)\cdots\phi(u_2)u_1 b \\ v_1\phi(v_2)\cdots\phi^{B-1}(v_B)\phi^{\kappa}(v_{B+1})\cdots\phi^{\kappa+\ell-(B+1)}(v_\ell). \quad (2)$$

Now, consider the derivation δ in $\widehat{G_a}$ leading to the word α defined in the beginning of the proof. This derivation can be decomposed as $a \stackrel{*}{\Longrightarrow} \beta \stackrel{*}{\Longrightarrow} \alpha$, where $\beta = \langle u_\ell\rangle_{i_\ell}\cdots\langle u_{B+1}\rangle_{i_{B+1}} b' \langle v_{B+1}\rangle_{i_{B+1}}\cdots\langle v_\ell\rangle_{i_\ell}$ where $b' \in [a]$. This means that we have a sub-derivation δ' of the form $b' \stackrel{*}{\Longrightarrow} \langle u_B\rangle_{i_B}\cdots\langle u_1\rangle_{i_1} b\langle v_1\rangle_{i_1}\cdots\langle v_B\rangle_{i_B}$. We define another derivation in $\widehat{G_a}$ which produces first the word β as in δ, then iterates $\kappa - B$ times the rule $b' \to \langle\epsilon\rangle b'\langle\epsilon\rangle$ (recall that $b' \to b'$ must exist in G_a since the original substitution ϕ is normal), and finally applies the rules of δ'. This derivation produces a nonterminal word α' of the form

$$\langle u_\ell\rangle_{i_{\kappa+\ell-B}}\cdots\langle u_{B+1}\rangle_{i_{\kappa+1}}\langle\epsilon\rangle^{\kappa-B}\langle u_B\rangle_{i_B}\cdots\langle u_1\rangle_{i_1} b \\ \langle v_1\rangle_{i_1}\cdots\langle v_B\rangle_{i_B}\langle\epsilon\rangle^{\kappa-B}\langle v_{B+1}\rangle_{i_{\kappa+1}}\cdots\langle v_\ell\rangle_{i_{\kappa+\ell-B}}.$$

Then, we deduce from (2) and Lemma 6 that $\sigma \in \phi^*(a)$. This completes the proof of Lemma 10. □

Lemma 11. *If there is a derivation in $\widehat{G_a}$ of length ℓ from a to some terminal word $\langle u_\ell\rangle_{i_\ell}\cdots\langle u_2\rangle_{i_2}\langle u_1\rangle_{i_1}\langle\langle w\rangle\rangle_i\langle v_1\rangle_{i_1}\langle v_2\rangle_{i_2}\cdots\langle v_\ell\rangle_{i_\ell}$, then*

$$\phi^{\geq B}(u_\ell)\cdots\phi^{\geq B}(u_B)\phi^{B-1}(u_{B-1})\cdots\phi(u_1)w \\ \phi(v_1)\cdots\phi^{B-1}(v_{B-1})\phi^{\geq B}(v_B)\cdots\phi^{\geq B}(v_\ell) \subseteq \phi^*(a).$$

Proof. Similar to the proof of Lemma 10. □

We are now able to give a construction for $\phi^*(a)$: Consider the nonterminal words $\langle u_\ell\rangle_{i_\ell}\cdots\langle u_1\rangle_{i_1} b\langle v_1\rangle_{i_1}\cdots\langle v_\ell\rangle_{i_\ell}$ generated by the grammar $\widehat{G_a}$. They form an effective linear context-free language. By a regular transduction, we can decorate the terminal symbols in these words by indices in $\{0,1,...,B\}$ in the following manner: each nonterminal symbol $\langle u_j\rangle_{i_j}$ or $\langle v_j\rangle_{i_j}$ is decorated by $j-1$ if $j < B$, otherwise by B. This yields another effective context-free linear language. Now, by induction hypothesis, we know that for every symbol b in T_a, the set $\phi^*(b)$ is effective, and hence, the set $\phi^{\geq B}(u)$ is effective, for any $u \in T_a^*$. Therefore, we can use another transduction to substitute the finite set $\phi^j(u)$ to

each nonterminal symbol $\langle u \rangle_i$ decorated by an index $j < B$, and to substitute the (effective) regular set $\phi^{\geq B}(u)$ to each $\langle u \rangle_i$ decorated by the index B. We get in this manner a new linear context-free language, we call L_1.

We apply a similar process to terminal words $\langle u_\ell \rangle_{i_\ell} \cdots \langle u_1 \rangle_{i_1} \langle\langle w \rangle\rangle_i \langle v_1 \rangle_{i_1} \cdots \langle v_\ell \rangle_{i_\ell}$ generated by $\widehat{G_a}$. In this case, the central symbol $\langle\langle w \rangle\rangle_i$ is decorated by index 0, and each nonterminal symbol $\langle u_j \rangle_{i_j}$ or $\langle v_j \rangle_{i_j}$ is decorated by j if $j < B$, otherwise by B. As in the previous cases, we obtain an effective linear language. Let us call this language L_2^0. Then, we repeat this process, but this time, we move up all the indices by one. In this manner, the regular substitution performed after the indexing step replaces $\langle\langle w \rangle\rangle_i$ by $\phi(w)$, all the $\langle u_j \rangle_{i_j}$'s such that $j < B - 1$ by $\phi^{j+1}(u_j)$, and all the $\langle u_j \rangle_{i_j}$'s such that $j \geq B - 1$ by $\phi^B(u_j)$. Let us call L_2^1 the so obtained linear language. We repeat this process B times. The union of all the languages L_2^j is an effective linear language L_2.

Lemma 12. *The language $L_1 \cup L_2$ is precisely $\phi^*(a)$ and it is effectively regular.*

Proof. By Lemma 10 and Lemma 11, we know that $L_1 \cup L_2 \subseteq \phi^*(a)$. Let us consider the reverse inclusion. Let σ be a word in $\phi^*(a)$. There are two cases to consider:

First, suppose that σ is a nonterminal word (i.e., of the form ubv where $b \in [a]$). Then, by Lemma 7 we know that $\sigma \in L_1$.

Consider now that σ is a terminal word (i.e. $\sigma \in T_a^*$). Then, by Lemma 9, there exists an intermediate terminal word σ' such that $\sigma' \in L_2^0$ and $\sigma \in \phi^n(\sigma)$ for some $n \geq 0$. It can be seen that, if $n < B$, then $\sigma \in L_2^n$, otherwise $\sigma \in L_2^B$. In both cases, $\sigma \in L_2$. This completes the proof of the fact that $L_1 \cup L_2 = \phi^*(a)$.

As for the effective regularity of $L_1 \cup L_2$, we know that this language is an effective context-free language, but we know also that it must be a downward closed language since ϕ is downward closed. Then, the result follows from Proposition 1. □

Lemma 12 completes the case where a is a linear symbol in the inductive step of the proof of Lemma 5. Hence, Lemma 5 is proved. □

Proposition 2 follows from Lemma 5 and Theorem 1. □

Theorem 2 follows from Lemma 1, Lemma 2, and Proposition 2. □

Remark 2. By proving the result above (in fact from Lemma 2, and Proposition 2), we have proved that the set of words generated by any weak $0L$-system is an SRE effective set.

We can adapt the proof presented above in order to show that the set of subwords of the language generated by any $0L$-system is an effective SRE set.

Theorem 3. *For every context-free language L, and every $0L$-system (R, φ_R), the set $\varphi_R^*(L)\downarrow$ is effectively SRE representable.*

4 Unconstructibility Results

We show in this section that the weak cyclic closure is unconstructible in general, even though we know that it is always regular. For that, we use known results on *lossy channel systems* (LCS for short). Such a system consists of a finite-state machine operating on a single FIFO buffer which is unbounded and lossy in the sense that it can nondeterministically lose messages (see [AJ96] for a formal definition).

The set of reachable configurations of any LCS from any given configuration is downard closed. Thus, we can deduce from Theorem 1 that this set is SRE definable. However, from [May00] we know the following:

Lemma 13. *For any LCS $\mathcal{L}$ and any configuration γ, the set of reachable configuration of $\mathcal{L}$ starting from γ is in general not computable.*

We use this lemma to establish that the weak cyclic closure is not constructible already for 1-dim rewriting systems. For that, given a lossy channel system $\mathcal{L}$, we derive a 1-dim cyclic rewriting system $\mathcal{R}$ which "simulates" $\mathcal{L}$. This simulation can be done as soon as we allow rules with two symbols in their left hand sides.

Theorem 4. *There is no algorithm which can construct the set $R^*_{wc}(\epsilon)$ for any given finite set of 1-dim rewriting rules R.*

The actions of a LCS can be straightforwardly encoded as rules of a 2-dim context-free cyclic rewriting system. Hence, we have the following fact.

Theorem 5. *There is no algorithm which can construct the set $R^*_{wc}(\epsilon, \epsilon)$ for any given finite set of 2-dim context-free rewriting rules R.*

5 Constructibility for n-Dim Context-Free Rings

We show in this section that the weak cyclic closure is constructible for a special kind of 2-dim context-free rewriting systems.

A set of n-dim rewriting rules R is a *ring* if, for every rule $r : (x_1, \ldots, x_n) \mapsto (y_1, \ldots, y_n)$ in R, $\exists i \in \{1, \ldots, n\}$. $\forall j \neq i$. $x_j = \epsilon$ and $\forall j \neq (i+1) \bmod n$. $y_j = \epsilon$. Hence, r is either of the form $(\epsilon, \ldots, \epsilon, x_i, \epsilon, \ldots, \epsilon) \mapsto (\epsilon, \ldots, \epsilon, y_{i+1}, \epsilon, \ldots, \epsilon)$ or of the form $(\epsilon, \ldots, \epsilon, x_n) \mapsto (y_1, \epsilon, \ldots, \epsilon)$. Notice that every set of 1-dim rewriting rules is a 1-dim ring and vice-versa.

We consider hereafter n-dim context-free rings, i.e., the x_i's are symbols in Σ. Intuitively, the rules of these systems correspond to actions of FIFO-channel systems where a symbol x is received from some channel of index i, and a word y is sent to the channel of index $(i + 1) \bmod n$.

We can show that the computation of the weak cyclic closure for rings can be reduced to the problem of computing the weak cyclic closure of 1-dim context free rewriting systems.

Theorem 6. *For every n-dim context-free ring R, and every n-dim language L which is a product of n context-free languages (effectively described by context-free grammars), the set $R^*_{wc}(L)$ is effectively SRE representable.*

References

[AAB99] Parosh Aziz Abdulla, Aurore Annichini, and Ahmed Bouajjani. Algorithmic verification of lossy channel systems: An application to the bounded retransmission protocol. In *TACAS'99*. LNCS 1579, 1999.

[ABJ98] Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-fly analysis of systems with unbounded, lossy fifo channels. In *CAV'98*. LNCS 1427, 1998.

[AJ96] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.

[BBK93] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the ACM*, (40):653–682, 1993.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In *CAV'96*. LNCS 1102, 1996.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *SAS'97*. LNCS 1302, 1997.

[BH97] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. In *ICALP '97*. LNCS 1256, 1997.

[Bur98] O. Burkart. Queues as processes. *Electrnic Notes in Theoretical Computer Science*, 18, 1998.

[CFI96] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable Channels Are Easier to Verify Than Perfect Channels. *Information and Computation*, 124(1):20–31, 1996.

[CHS92] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. In *CONCUR '92*. LNCS, 1992.

[Cou91] B. Courcelle. On constructing obstruction sets of words. *EATCS*, 44:178–185, June 1991.

[CVWY90] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. In *CAV'90*, 1990.

[GH94] J.F. Groote and H. Hüttel. Undecidable equivelences for basic process algebra. *Information and Computation*, 1994.

[GvdP93] J.F. Groote and J. van de Pol. A bounded retransmission protocol for large data packets. Tech. report, Dept. of Philosophy, Utrecht University, Oct. 1993.

[Hol91] G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[May00] R. Mayr. Undecidable problems in unreliable computations. In *LATIN'2000*. LNCS 1776, 2000.

[RS80] G. Rozenberg and A. Salomaa. *The Mathematical Theory of L-systems*. Academic Press, 1980.

[Sti96] C. Stirling. Decidability of bisimulation equivalence for normed pushdown processes. In *CONCUR'96*. LNCS 1119, 1996.

Model Checking of Unrestricted Hierarchical State Machines

Michael Benedikt[1], Patrice Godefroid[1], and Thomas Reps[2]

[1] Bell Laboratories, Lucent Technologies, {benedikt,god}@bell-labs.com
[2] University of Wisconsin, reps@cs.wisc.edu

Abstract. Hierarchical State Machines (HSMs) are a natural model for representing the behavior of software systems. In this paper, we investigate a variety of model-checking problems for an extension of HSMs in which state machines are allowed to call each other recursively.

1 Introduction

Hierarchical State Machines (HSMs) are finite-state machines whose states themselves can be other machines. HSMs form the basis of several commercial modeling languages, such as StateCharts, ObjecTime, and UML. Various verification problems for HSMs without recursion have been studied in [5,4,3].

In this paper, we investigate an extension of HSMs in which machines are allowed to call each other recursively. Such "unrestricted" HSMs are strictly more expressive than the previously-studied HSM model since HSMs with recursion can model classes of infinite-state systems. For instance, unrestricted HSMs can be used to model the control-flow graphs of procedures in programming languages such as C. Unrestricted HSMs are therefore a natural model for reasoning about the abstract behavior of reactive software programs.

We study several verification problems for unrestricted HSMs. First, we define several classes of unrestricted HSMs (or HSMs for short), and establish correspondence theorems with previously-existing classes of infinite-state systems. Specifically, we show that "single-exit" HSMs, i.e., HSMs composed exclusively of machines each with a single exit state, have the same expressiveness as context-free processes, while general "multiple-exit" HSMs have the same expressiveness as pushdown processes. From these correspondence theorems and known verification results for context-free and pushdown systems, we immediately obtain algorithms and complexity bounds for various verification problems on HSMs.

We then show how some of the above results can be improved via new verification algorithms. We present an LTL model-checking algorithm for unrestricted HSMs. This algorithm shows that LTL model checking for single-entry multiple-exit HSMs (i.e., HSMs composed of machines each with a single entry state, but possibly multiple exit states) can be solved in time linear in the size of the HSM, instead of cubic time as previously known. This implies that the reachability and cycle-detection problems can be solved in linear time for single-entry HSMs.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 652–666, 2001.

We also present a new model-checking algorithm for the logic CTL* and single-exit HSMs. The algorithm runs in time linear in the size of the HSM, instead of quadratic time, the best previously-known upper bound. Due to the correspondence results mentioned above, this algorithm also provides an improved upper bound for CTL* model checking of context-free processes.

2 Unrestricted Hierarchical State Machines

A (flat) *Kripke structure* K over a set of atomic propositions P is a tuple (S, R, L), where S is a (possibly infinite) set of states, $R \subseteq S \times S$ is a transition relation, and $L : S \mapsto 2^P$ is a labeling function that associates with each state the set of atomic propositions that are true in that state.

In this paper, we consider *unrestricted hierarchical state machines* (HSMs) M over a set P of atomic propositions; these consist of a set of *component structures* $\{M_1, \ldots, M_n\}$, where each of the M_i has

- A nonempty finite set N_i of *nodes*.
- A finite set B_i of *boxes*.
- A nonempty subset I_i of N_i, called the *entry-nodes* of N_i.
- A nonempty subset O_i of N_i, called the *exit-nodes* of N_i.
- A labeling function $X_i : N_i \mapsto 2^P$ that labels each node with a subset of P.
- An indexing function $Y_i : B_i \mapsto \{1, \ldots, n\}$ that maps each box of M_i to the index j of some structure M_j.
- A set C_i of pairs of the form (b, e), where b is a box in B_i and e is an entry-node of M_j with $j = Y_i(b)$, called the *call-nodes* of B_i.
- A set R_i of pairs of the form (b, x), where b is a box in B_i and x is an exit-node of M_j with $j = Y_i(b)$, called the *return-nodes* of B_i.
- An edge relation E_i. Each edge in E_i is a pair (u, v) such that (1) u is either a node in N_i or a return-node in R_i, and (2) v is either a node in N_i or a call-node in C_i.

M_1 is called the top-level structure of M. The above definition is essentially that of Alur and Yannakakis [5]; however, we permit component structures to call each other recursively. An example of an unrestricted HSM is shown in Fig. 1.

To simplify notation in what follows, we assume that the sets I_i and O_i are all pairwise disjoint, as are the sets C_i and R_i. (Note that C_i and R_i are technically not part of N_i.) To be able to find all of the boxes that call a given component machine j, we define $callers(j) = \{(b, i) \mid Y_i(b) = j\}$.

An HSM M is called *single-entry* if every structure M_i in M has exactly one entry-node (i.e., $\forall 1 \leq i \leq n : |I_i| = 1$). An HSM M is called *single-exit* if every structure M_i in M has exactly one exit-node (i.e., $\forall 1 \leq i \leq n : |O_i| = 1$).

Each structure M_i can be associated with an ordinary Kripke structure, denoted $K(M_i)$, by recursively substituting each box $b \in B_i$ by the structure M_j with $j = Y_i(b)$. Since we allow state machines to call each other recursively, the expanded structure $K(M_i)$ can be infinite. A state of the expanded Kripke structure $K(M)$ is defined by a node and a finite sequence of boxes that specify

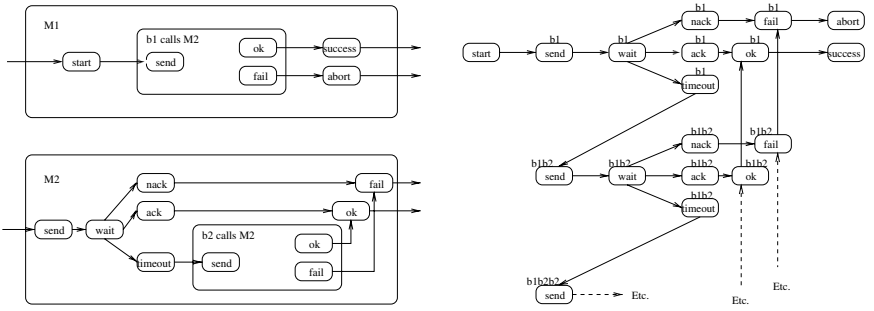


Fig. 1. An example of an unrestricted HSM (left) and its expansion (right). The top-level structure M_1 has one box, which calls structure M_2. Structure M_2 models an attempt to send a message; if no positive or negative acknowledgment is received, a timeout occurs and a recursive call to M_2 is performed.

the context. Formally, the *expansion* $K(M)$ *of an HSM* M is the Kripke structure (S, R, L) defined as follows:

- $S \subseteq \bigcup_{i=1}^{n} N_i \times (\bigcup_{i=1}^{n} B_i)^*$.
- R is the set of transitions $((v, w), (v', w'))$ that satisfy any of the following:
  - $(v, v') \in E_i$, $v, v' \in N_i$ and $w = w'$.
  - $(v, (b', e')) \in E_i$, $v \in N_i$, $v' = e'$, and $w' = wb'$.
  - $((b, x), v') \in E_i$, $v = x$, $v' \in N_i$, and $w = w'b$.
  - $((b, x), (b', e')) \in E_i$, $v = x$, $v' = e'$, and $w' = w''b'$ with $w = w''b$.
- $L : S \mapsto 2^P$ is defined by $L((v, w)) = X_i(v)$ with $v \in N_i$.

The (infinite) expansion $K(M_1)$ of the HSM of Fig. 1 is shown on the right of the figure, where the finite sequence of boxes corresponding to each state is indicated on top of the state when it is nonempty (e.g., the state "(send,b1b2)" is depicted as the state "send" labeled with "b1b2"). We will write $K(M)$ to denote the expansion of the top-level structure M_1 of an HSM M.

3 Expressiveness of Unrestricted HSMs

Unrestricted HSMs are closely related to several existing models for infinite-state systems, namely context-free grammars and pushdown automata. In this section, we compare the expressiveness and concision of these models. We also compare the expressiveness of the four classes of unrestricted HSMs defined in the previous section, namely single-entry single-exit, single-entry multiple-exit, multiple-entry single-exit, and multiple-entry multiple-exit HSMs.

Since we are interested in the temporal behavior of systems, our comparison of expressiveness is based on the existence of bisimulation relations between the Kripke structures corresponding to the expansions of these different classes of models. Given two Kripke structures $M_1 = (S_1, R_1, L_1)$ and $M_2 = (S_2, R_2, L_2)$, a binary relation $\mathcal{B} \subseteq S_1 \times S_2$ is a *bisimulation relation* if $(s_1, s_2) \in \mathcal{B}$ implies: (1) $L_1(s_1) = L_2(s_2)$, (2) if $(s_1, s_1') \in R_1$, then there is some $s_2' \in S_2$ such that

$(s_2, s_2') \in R_2$ and $(s_1', s_2') \in \mathcal{B}$, and (3) if $(s_2, s_2') \in R_2$, then there is some $s_1' \in S_1$ such that $(s_1, s_1') \in R_1$ and $(s_1', s_2') \in \mathcal{B}$. Two states s_1 and s_2 are *bisimilar*, denoted $s_1 \sim s_2$, if they are related by some bisimulation relation. By extension, we say that two Kripke structures M_1 and M_2 are bisimilar if $\forall s_1 \in S_1 : \exists s_2 \in S_2 : s_1 \sim s_2$ and $\forall s_2 \in S_2 : \exists s_1 \in S_1 : s_1 \sim s_2$.

Obviously, any multiple-entry machine with k entry-nodes can be replaced by k machines, each with a single entry-node. Therefore, the expressiveness of single-entry and multiple-entry HSMs is the same, although multiple-entry HSMs can be more concise than their equivalent single-entry HSM. In contrast, we show in the remainder of this section that single-exit and multiple-exit HSMs have different expressivenesses. Indeed, single-exit HSMs have the same expressiveness as context-free processes while multiple-exit HSMs have the same expressiveness as pushdown processes.

An *alphabetic labeled rewrite system* [9] is a triple $\mathcal{R} = (V, Act, R)$ where V is an alphabet, Act is a set of labels, and $R \subset V \times Act \times V^*$ is a finite set of rewrite rules. The *prefix rewriting relation* of $\mathcal{R}$ is defined by $\mapsto_{\mathcal{R}} = \{(uw, a, vw) | (u, a, v) \in R, w \in V^*\}$. The labeled transition graph $T_{\mathcal{R}} = (V^*, Act, \mapsto_{\mathcal{R}})$ is called the *prefix transition graph* of $\mathcal{R}$. Since the leftmost derivation graph of any context-free grammar [14] is the prefix transition graph of an alphabetic rewrite system [9], such prefix transition graphs are sometimes called *context-free processes*. For purposes of comparison with HSMs, we define the *expansion* of $\mathcal{R}$ as the (possibly infinite) Kripke structure $K(\mathcal{R})$ defined as follows: a state of $K(\mathcal{R})$ is a pair $(a, w) \in Act \times V^*$ such that $(v, a, w) \in \mapsto_{\mathcal{R}}$ for some $v \in V^*$; a transition of $K(\mathcal{R})$ is a pair $((a, w), (a', w'))$ such that (w, a', w') is in $\mapsto_{\mathcal{R}}$; the label of state (a, w) is a. We can now prove the following theorem:

Theorem 1. *For any alphabetic labeled rewrite system $\mathcal{R}$, one can construct in linear time a single-exit HSM M such that $K(\mathcal{R})$ and $K(M)$ are bisimilar.*

The converse of the previous theorem also holds:

Theorem 2. *For any multiple-entry single-exit HSM M, one can construct in linear time an alphabetic labeled rewrite system $\mathcal{R}$ such that $K(M)$ and $K(\mathcal{R})$ are bisimilar.*

We now establish a similar correspondence between multiple-exit HSMs and pushdown processes. A *pushdown automaton* (e.g., [14]) is a tuple $A = (Q, Act, \Gamma, \delta, q_0)$ where Q is a finite set of states, Act is an alphabet called the input alphabet, Γ is a set of stack symbols, $q_0 \in Q$ is the initial state, and δ is a mapping from $Q \times Act \times \Gamma$ to finite subsets of $Q \times \Gamma^*$. The initial configuration of the system is (q_0, ϵ). The *expansion* of A is the (possibly infinite) Kripke structure $K(A)$ defined by the expansion of the prefix rewriting relation $\mapsto_\delta \subseteq (Q \times \Gamma^*) \times Act \times (Q \times \Gamma^*)$ itself defined by $\mapsto_\delta = \{((q, Z\gamma), a, (q', \beta\gamma)) | (q', \beta) \in \delta(q, a, Z), \gamma \in \Gamma^*\}$. We call such a Kripke structure a *pushdown process*. We have the following:

Theorem 3. *For any pushdown automaton A, one can construct in linear time a multiple-exit HSM M such that $K(A)$ and $K(M)$ are bisimilar.*

Conversely, the following theorem also holds:

Class of HSM	Reachability	Cycle Detection	LTL	CTL	CTL*
Restricted Single-exit	Linear	Linear	Linear	Linear	
Restricted Multiple-exit	Linear	Linear	Linear	PSPACE	
Unrestricted Single-exit	Linear	Linear	Linear	Linear	Quadratic
Unrestricted Multiple-exit	Cubic	Cubic	Cubic	EXPTIME	EXPTIME

Fig. 2. Complexity bounds derived from Sect. 3 and previously known results. (Complexity bounds are given in terms of the size of the HSM.)

Theorem 4. *For any multiple-entry multiple-exit HSM M, one can construct in linear time a pushdown automaton A such that $K(M)$ and $K(A)$ are bisimilar.*

Since it is known [9] that there exist pushdown processes that are not bisimilar to any context-free processes, we obtain the following result:

Theorem 5. *There exist multiple-exit HSMs whose expansion is not bisimilar to the expansion of any single-exit HSM.*

4 Complexity of Verification Problems for HSMs

In this section, we discuss the complexity of five verification problems for unrestricted HSMs: the reachability problem, the cycle-detection problem, and the model-checking problems for the logics LTL, CTL, and CTL* [10]. Given an unrestricted HSM M and a set $T \subseteq \bigcup_{i=1}^{n} N_i$ of distinguished nodes, the *reachability problem* is the problem of determining whether some state (v, w) of $K(M)$, with $v \in T$, is reachable from some initial state (v_0, ϵ), with $v_0 \in I_1$. Given M and T, the *cycle-detection problem* is to determine whether there exists some state $(v, w)$of $K(M)$, with $v \in T$, such that (i) (v, w) is reachable from some initial state (v_0, ϵ), with $v_0 \in I_1$, and (ii) (v, w) is reachable from itself.

Since restricted HSMs are special cases of unrestricted HSMs, it is worth reviewing some of the results presented in [5] for the restricted case. Lines 2 and 3 of Fig. 2 summarize the results of [5] concerning the complexity of the verification problems considered here, except for CTL* model checking, which was not discussed in [5]. Complexity bounds are given in terms of the size of the restricted HSM; in the case of LTL and CTL model checking, this means the size of the formula is fixed. (It is also shown in [5] that, for any fixed restricted HSM, CTL model checking is PSPACE-complete in the size of the formula.)

Thanks to the correspondence theorems established in the previous section, we can obtain algorithms and complexity bounds for the verification of unrestricted HSMs from previously existing algorithms and bounds for the verification of context-free and pushdown processes.

For single-exit unrestricted HSMs, Theorem 2 implies that model checking for single-exit HSMs can be reduced to model checking for context-free processes. Since context-free processes can be viewed as pushdown processes defined by pushdown automata with only one state [7,20], and since LTL model checking for one-state pushdown automata can be solved in time linear in the size of the pushdown automaton [12,13], LTL model checking for single-exit HSMs can be

Class of Unrestricted HSM	Reachability	Cycle detection	LTL	CTL	CTL*
Multiple-entry Single-exit	Linear	Linear	Linear	Linear	*Linear*
Single-entry Multiple-exit	*Linear*	*Linear*	*Linear*	EXPTIME	EXPTIME
Multiple-entry Multiple-exit	Cubic	Cubic	Cubic	EXPTIME	EXPTIME

Fig. 3. Improved complexity bounds for unrestricted HSMs. The improved bounds obtained in Sects. 5 and 6 are highlighted in italic.

solved in time linear in the size of the HSM. This also implies a linear-time algorithm for the reachability and cycle-detection problems. A linear-time algorithm for CTL model checking for single-exit HSMs can be derived from the CTL model-checking algorithm for context-free processes given in [7]. Finally, since the μ-calculus model-checking algorithm of [8] for context-free processes runs in quadratic time for formulae in the second level of the μ-calculus alternation hierarchy, which is known to contain CTL* [11], CTL* model checking for single-exit HSMs can be solved in time quadratic in the size of the HSM.

In the case of multiple-exit unrestricted HSMs, Theorem 4 implies that model checking for multiple-exit HSMs can be reduced to model checking for pushdown processes. Since LTL model checking for pushdown automata can be solved in time cubic in the size of the pushdown automaton [13,12], LTL model checking for multiple-exit HSMs can be solved in time cubic in the size of the HSM. Moreover, a cubic-time algorithm for the reachability and cycle-detection problems can easily be derived from this LTL model-checking algorithm. Since CTL model checking for pushdown processes is EXPTIME-hard [20] and since CTL is contained in the alternation-free μ-calculus for which the model-checking problem can be solved with the exponential-time algorithm presented in [6], we can deduce from Theorems 3 and 4 that the CTL model-checking problem for multiple-exit HSMs is EXPTIME-complete in the size of the HSM. Similarly, the exponential-time model-checking algorithm given in [8] for pushdown processes and the full μ-calculus, which contains CTL*, and the EXPTIME-hardness result of [20] imply that the CTL* model-checking problem for multiple-exit HSMs is also EXPTIME-complete in the size of the HSM. The bottom two lines of Fig. 2 summarize the results obtained from the foregoing discussion.

In the remainder of this paper, we present two improvements to the results listed in Fig. 2. First, in Sect. 5, we present an LTL model-checking algorithm for unrestricted HSMs, and analyze the complexity of this algorithm. We then show that LTL model checking for single-entry multiple-exit HSMs can be solved with this algorithm in time *linear* in the size of the HSM, instead of *cubic* time. This implies that the reachability and cycle-detection problems can also be solved in linear time for single-entry HSMs. Second, in Sect. 6, we present a new CTL* algorithm for single-exit HSMs that runs in time *linear* in the size of the HSM, instead of *quadratic* time. Improved complexity bounds that take into account these two new results are listed in Fig. 3.

5 LTL Model Checking

Following the automata-theoretic approach to model checking [19], a model-checking procedure for a formula ϕ of linear-time temporal logic can be obtained by (1) building a finite-state Büchi automaton $A_{\neg\phi}$ that accepts exactly all the infinite words satisfying the formula $\neg\phi$, (2) creating a product automaton for $A_{\neg\phi}$ and the system to be verified, and (3) checking if the language accepted by the product automaton is empty. To apply this procedure in our context, we define the product of a Büchi automaton $A_{\neg\phi}$ with an HSM[1] $M = \{M_1, \ldots, M_n\}$ to be a *Büchi-constrained HSM* $M' = \{M'_1, \ldots, M'_n\}$: M' is an HSM as defined earlier, where the labeling function encodes a Büchi acceptance condition. In particular, the nodes in node set N'_i of component structure M'_i are pairs (v, s), where $v \in N_i$ and s is a state of $A_{\neg\phi}$. Each box in B'_i is also a pair (b, s), where $b \in B_i$ and s is a state of $A_{\neg\phi}$, and such that $Y'_i((b, s)) = Y_i(b)$. Moreover, we have $C'_i = \{((b, s), (e, s))|(b, s) \in B'_i \text{ and } (b, e) \in C_i\}$ and $R'_i = \{((b, s), (x, s))|(b, s) \in B'_i \text{ and } (b, x) \in R_i\}$. Edges in the edge sets E'_i are of the form $(v, s) \to (v', s')$, such that there is an edge $v \to v'$ in E_i and a transition (s, ℓ, s') in $A_{\neg\phi}$, where $\ell \in 2^P$ agrees with the set of propositions true at v if $v \in N_i$, or else ℓ agrees with the set of propositions true at x if v is a return-node $(b, x) \in R_i$.

We define the labeling function X' on nodes (v, s) of M' such that $X'((v, s))$ is *true* if s is an accepting state of $A_{\neg\phi}$, and *false* otherwise. Let T denote the set of nodes of M' where X' is *true*. The LTL model-checking problem for an HSM M and formula ϕ is thus reduced to checking whether there exists an infinite sequence w of states in $K(M')$ such that w passes through a node in T infinitely often. (Note that $K(M') = K(M) \times A_{\neg\phi}$, where $\times$ denotes the traditional definition of the product of a Kripke structure with a Büchi automaton.)

The latter problem can in turn be reduced to a graph-theoretic problem expressed in terms of the finite graph $G(M')$ whose nodes are the nodes of M' and whose edges are the edges of M' plus the set $\mathit{CallEdges}(M') \cup \mathit{ReturnEdges}(M')$, where $\mathit{CallEdges}(M') = \{((b, e), e) \mid e \in I'_i, b \in B'_j, Y'_j(b) = i\}$ and $\mathit{ReturnEdges}(M') = \{(x, (b, x)) \mid x \in O'_i, b \in B'_j, Y'_j(b) = i\}$. This graph finitely and completely represents $K(M')$, while making explicit how behaviors of component structures M'_i can be combined with calls and returns between component structures: every possible execution sequence in $K(M')$ is represented by a path in $G(M')$. However, not all paths in $G(M')$ represent execution paths of $K(M')$: a path in $G(M')$ corresponds to a path in $K(M')$ if, when a call finishes, the path in $G(M')$ returns to a return-node of the invoking box. The following definition characterizes the paths of $G(M')$ that correspond to executions of $K(M')$.

Definition 6. Give each box in M' a unique index in the range $1 \ldots |B|$, where $|B|$ is the total number of boxes in M'. For each box b, label the associated call-edges and return-edges with the symbols "$(_b$" and "$)_b$", respectively; label all other edges with "e". A path in $G(M')$ is called a *Bal-path* (resp. *UnbalLeft-path*) iff the word formed by concatenating, in order, the symbols on the path's edges is in the language $L(\mathit{Bal})$ (resp. $L(\mathit{UnbalLeft})$), defined as follows:

[1] As usual in this context, we assume for technical convenience that every node in N_i has an E_i successor.

```
function CompSummaryEdges(M: HSM, T ⊆ ⋃_{i=1}^{n} N_i) returns set of pairs (edge,Bool)
[1]   PathEdges, SummaryEdges, WorkList: set of pairs (edge,Bool)

      procedure Propagate(e → v: edge, B: Bool)
[2]     if there is no pair of the form (e → v, B') in PathEdges then
[3]      Insert (e → v, B) into PathEdges
[4]      Insert (e → v, B) into WorkList
[5]     else if (e → v, B') ∈ PathEdges ∧ B = true ∧ B' = false then
[6]      PathEdges := (PathEdges − {(e → v, B')}) ∪ {(e → v, B)}
[7]      WorkList := (WorkList − {(e → v, B')}) ∪ {(e → v, B)}
[8]     fi
      end

[9]   PathEdges := ∅; SummaryEdges := ∅; WorkList := ∅
[10]  for each entry-node e of some I_i, for 1 ≤ i ≤ n do Propagate(e → e, e ∈ T) od
[11]  while WorkList ≠ ∅ do
[12]    Select and remove a pair (e → v, B) from WorkList
[13]    switch v
[14]     case v = (b, e') ∈ C_i: /* v is a call-node */
[15]      for each (b, x) such that ((b, e') → (b, x), B') ∈ SummaryEdges do
[16]       Propagate(e → (b, x), B ∨ B')
[17]      od
[18]     end case
[19]     case v = x ∈ O_i: /* v is an exit-node */
[20]      for each pair (b, j) ∈ callers(i) do /* b ∈ B_j and Y_j(b) = i */
[21]       if there is no pair of the form ((b, e) → (b, x), B') in SummaryEdges then
[22]        Insert ((b, e) → (b, x), B) into SummaryEdges
[23]       else if ((b, e) → (b, x), B') ∈ SummaryEdges ∧ B = true ∧ B' = false then
[24]         SummaryEdges := (SummaryEdges − {((b, e) → (b, x), B')}) ∪ {((b, e) → (b, x), B)}
[25]       fi
[26]       for each e' ∈ I_j such that (e' → (b, e), B'') ∈ PathEdges do
[27]        Propagate(e' → (b, x), B ∨ B'')
[28]       od
[29]      od
[30]     end case
[31]     default : /* v ∈ (N_i − O_i) ∪ R_i, i.e., v is not a call-node or an exit-node */
[32]      for each v' such that v → v' ∈ E_i do Propagate(e → v', B ∨ (v' ∈ T)) od
[33]     end case
[34]    end switch
[35]  od
[36]  return(SummaryEdges)
```

Fig. 4. An algorithm for computing summary-edges for a Büchi-constrained HSM M with Büchi acceptance condition T.

$Bal \rightarrow Bal \quad Bal$ $\mid (_j \quad Bal \quad)_j \qquad 1 \leq j \leq \lvert B \rvert$ $\mid e$ $\mid \epsilon$	$UnbalLeft \rightarrow UnbalLeft \quad (_j \quad Bal \qquad 1 \leq j \leq \lvert B \rvert$ $\mid Bal$

LTL model checking is carried out directly on the Büchi-constrained product-HSM by means of the two-phase algorithm presented in Figs. 4 and 5. In the first phase, the dynamic-programming algorithm CompSummaryEdges, shown in Fig. 4 is applied to an HSM[2] M with Büchi acceptance condition T to create a set of *summary-edges*. Each summary-edge represents a Bal-path between a call-node and a return-node, where the two nodes are associated with the same box. More precisely, CompSummaryEdges creates the set $SummaryEdges$, which consists of pairs of the form $((b, e) \rightarrow (b, x), B)$. Summary-edge $((b, e) \rightarrow (b, x), B)$ indicates that (i) there exists a Bal-path from e to x, and (ii) if Boolean value

[2] Henceforth, we drop prime symbols ($'$) on components of Büchi-constrained HSMs.

```
function ContainsTCycle(M: HSM, T ⊆ ⋃_{i=1}^{n} N_i) returns a set of nodes
[1]  SummaryEdges = CompSummaryEdges(M, T)
[2]  G = (⋃_{i=1}^{n} N_i ∪ C_i ∪ R_i, ⋃_{i=1}^{n} E_i ⋃ CallEdges(M) ⋃ SummaryEdges)
[3]  SCCSet = FindSCCs(G, I_1) /* I_1 is the set of roots of the depth-first search */
[4]  for each non-trivial SCC (Nodes, Edges) ∈ SCCSet do
[5]   if (Nodes ∩ T ≠ ∅) or (∃((b, e) → (b, x), B) ∈ Edges : B = true) then
[6]    return(Nodes)
[7]   fi
[8]  od
[9]  return(∅)
```

Fig. 5. An algorithm for detecting T-cycles.

B is *true*, then there exists such a path that passes through at least one node in T. In addition to tabulating summary-edges, CompSummaryEdges builds up the set *PathEdges*: a *path-edge* $(e \to v, B)$ in *PathEdges* indicates the existence of a *Bal*-path from an entry-node $e \in I_i$ of component structure M_i to v, where $v \in N_i \cup C_i \cup R_i$. As with summary-edges, the Boolean value B records whether the *Bal*-path summarized by the edge traverses at least one node in T.

It is possible to make two improvements to CompSummaryEdges: first, path-edges in each component structure can be "anchored" at exit-nodes rather than at entry-nodes, and path-edges can be "grown" backwards rather than forwards (a technique also used in [15]); second, path-edges in component structures M_i where $|O_i| < |I_i|$ can be anchored at exit-nodes (and path-edges grown backwards), whereas in other component structures the path-edges can be anchored at entry-nodes (and path-edges grown forwards). Henceforth, we mean the latter version whenever we refer to CompSummaryEdges in what follows.

The second phase of the model-checking algorithm consists of lines [2]–[8] of function ContainsTCycle of Fig. 5. The goal of ContainsTCycle is to determine whether any component structure M_i contains a node n such that (i) n is reachable from some entry-node of I_1 along an *UnbalLeft*-path, and (ii) there is a non-empty cyclic *UnbalLeft*-path (which might merely be a cyclic *Bal*-path) that starts at n and contains a member of T. ContainsTCycle checks this condition by searching for (nontrivial) strongly connected components that are reachable from an entry-node of I_1 (line [3]) in a directed graph G that consists of the nodes and edges of all component structures of M, together with all of M's call-edges, plus the set of summary-edges computed by CompSummaryEdges (line [2]). The presence of call-edges and summary-edges is what allows information to be recovered from G about *UnbalLeft*-paths in M. The summary-edges permit ContainsTCycle to avoid having to explore *Bal*-paths between call-nodes and return-nodes of the same box, and, in particular, whether such nodes are connected by a *Bal*-path that contains a T node.

Theorem 7. *Given an HSM M and an LTL formula ϕ, $K(M)$ satisfies ϕ iff the algorithm of Fig. 5 applied to the Büchi-constrained HSM $M \times A_{\neg\phi}$ and its corresponding set T returns $\emptyset$.*

For any component structure M_i, the worst-case time complexity of CompSummaryEdges is equal to I_i, the number of entry-nodes of M_i (or O_i, if the number of exit-nodes is smaller), multiplied by the number of E_i edges plus

summary-edges in M_i. In the worst case, each box $b \in B_i$ can have a summary-edge from every call-node (b, e) to every return-node (b, x). Therefore, the contribution of M_i to the time complexity of CompSummaryEdges is bounded by $O(\min(I_i, O_i)\ (E_i + \Sigma_{b \in B_i} C_b R_b))$.

The size of the graph G computed by function ContainsTCycle is bounded by $O(\Sigma_{i=1}^{n}\ (E_i + \Sigma_{b \in B_i} C_b R_b + \Sigma_{b \in B_i} C_b)$, and finding the strongly connected components of G can be carried out in time linear in the size of G (e.g., see [1]). Thus, the total worst-case cost of ContainsTCycle is bounded by $O(\Sigma_{i=1}^{n}\ [\min(I_i, O_i)\ (E_i + \Sigma_{b \in B_i} C_b R_b)])$. In the case of single-entry, single-exit, and single-entry single-exit HSMs, this bound simplifies as follows:

Single-entry HSM	Single-exit HSM	Single-entry single-exit HSM
$O(E + R)$	$O(E + C)$	$O(E + B)$

where E, R, C, and B are the total numbers of ordinary edges, return-edges, call-edges, and boxes, respectively.

Note that the Büchi-constrained HSM $M' = M \times A_{\neg\phi}$ obtained by combining a single-entry (or single-exit) HSM M with the Büchi automaton $A_{\neg\phi}$ for an LTL formula ϕ will typically be multiple-entry (resp. multiple-exit). However, each component structure M'_i of M' will have at most $|S_{\neg\phi}|$ entry-nodes (resp. exit-nodes), where $|S_{\neg\phi}|$ is the number of states of the automaton $A_{\neg\phi}$. Therefore, for a fixed LTL formula ϕ, the term $\min(I'_i, O'_i)$ is bounded by the fixed constant $|S_{\neg\phi}|$. Thus, *for any fixed LTL formula ϕ, the LTL model-checking problem for an unrestricted HSM M that is single-entry or single-exit can be solved in time linear in the size of M.*

6 CTL* Model Checking for Single-Exit HSMs

In this section, we present a CTL* model-checking algorithm for single-exit HSMs that runs in time linear in the size of the HSM. The logic CTL* uses the temporal operators U (until), X (nexttime) and the existential path quantifier E, in addition to the operators $\neg$ (not) and $\vee$ (or). Two types of CTL^* formulas, path formulas and state formulas, are defined by mutual induction. Every atomic proposition is a state formula as well as a path formula. If p, q are both state formulas (resp., both path formulas) then $p \vee q$ and $\neg p$ are also state formulas (resp., path formulas). If p and q are path formulas, then pUq and Xp are also path formulas while Ep is a state formula. We use the abbreviation Fp for $trueUp$ and Gp for $\neg F \neg p$. Any CTL* state formula can be viewed as a boolean combination of existential formulas. An existential formula is either an atomic proposition or a CTL* state formula of the form $E\ \rho(p(\gamma_1) \leftarrow \gamma_1, \ldots, p(\gamma_n) \leftarrow \gamma_n)$, where ρ is an LTL formula over propositions $p(\gamma_1), \ldots, p(\gamma_n)$ in which each proposition $p(\gamma_i)$ is substituted by the corresponding CTL* state formula γ_i. (For a description of the semantics of CTL*, see [10].)

A key technical challenge is that the truth value of a temporal-logic formula in any state (v, w) of $K(M)$ may not only depend on the node v but also on the stack contents w. Fortunately, it is sufficient to consider only finitely many equivalence

function SPLIT(ϕ: LTL formula) **returns** Set of pairs ($\beta \in \mathrm{LTL}^+, \delta \in \mathrm{LTL}$)
[1] **if** $(\phi = P)$ **then return**$(\{(P, true)\})$
[2] **if** $(\phi = \phi_1 \vee \phi_2)$ **then return**$(\mathrm{SPLIT}(\phi_1) \cup \mathrm{SPLIT}(\phi_2))$
[3] **if** $(\phi = \neg\phi_1)$ **then return**$(\bigcup_{A \subseteq \mathrm{SPLIT}(\phi_1)} (\bigwedge_{(\beta,\delta)\in A} \neg\beta, \bigwedge_{(\beta,\delta)\in \mathrm{SPLIT}(\phi_1)-A} \neg\delta))$
[4] **if** $(\phi = Xp)$ **then return**$(\bigcup_{(\beta,\delta)\in \mathrm{SPLIT}(p)} (X\beta, \delta) \cup \{(exit, p)\})$
[5] **if** $(\phi = pUq)$ **then return**$(\bigcup_{\emptyset \neq A \subseteq \mathrm{SPLIT}(p)} (G \bigvee_{(\beta,\delta)\in A} \beta, \bigwedge_{(\beta,\delta)\in A} \delta \wedge pUq)$
$\cup \bigcup_{\emptyset \neq A \subseteq \mathrm{SPLIT}(p)} \bigcup_{(\beta',\delta')\in \mathrm{SPLIT}(q)} (\bigvee_{(\beta,\delta)\in A} (\beta U \beta'), \delta' \wedge \bigwedge_{(\beta,\delta)\in A} \delta))\}$

Fig. 6. The function SPLIT.

classes of possible stack contents, each equivalence class being represented by a *context*, as already observed in [7,8,5]. A context is a set of (here CTL*) formulas whose truth value at the exit node of a machine M_i determine the truth value of a formula ϕ at the root. The notion of context makes it possible to reason compositionally about HSMs.

Our algorithm exploits this idea and reduces the evaluation of a path formula ϕ on a sequence $w; w'$ of states, where w is finite while w' is infinite, to the evaluation of some formulas β and δ on the sequences w and w', respectively. We introduce a special atomic proposition *exit*, which holds only at the final state of a finite sequence w, and denote by LTL$^+$ the set of LTL formulas that can be expressed using this extended set of atomic propositions. The function SPLIT given in Fig. 6 specifies how the evaluation of an LTL formula ϕ can be decomposed as described above. (A conjunction over an empty set of formulas is defined to have the value *true*.) For instance, $w; w' \models Xp$ can be decomposed either into $w \models Xp$ and $w' \models true$ (for the case where $|w| > 1$), or into $w \models exit$ and $w' \models p$ (for the case where $|w| = 1$).

Given a set F of CTL* state formulas, let exists(F) denote the set of existential formulas that are elements or subformulas of elements of F. A set F of existential CTL* formulas is *closed* if, for every $\gamma = E\rho(p(\gamma_1) \leftarrow \gamma_1, \ldots, p(\gamma_n) \leftarrow \gamma_n) \in \mathrm{exists}(F)$, for every δ such that $(\beta, \delta) \in \mathrm{SPLIT}(\rho)$, $E\delta(p(\gamma_1) \leftarrow \gamma_1, \ldots, p(\gamma_n) \leftarrow \gamma_n)$ is also in F. The *closure* cl(ϕ) of a CTL* formula ϕ is the smallest closed set containing exists($\{\phi\}$). One can show, using properties of SPLIT, that cl(ϕ) is always finite for any CTL* formula ϕ. Let pd(ϕ) be the maximal nesting of path quantifiers (E) in a CTL* formula ϕ. Given a set F of CTL* formulas, let $\mathrm{pd}(F) = \max_{\gamma \in F}(\mathrm{pd}(\gamma))$. For ϕ with $\mathrm{pd}(\phi) \geq j$, let $\mathrm{cl}^{\leq j}(\phi)$ be the elements of cl(ϕ) with at most j nested path quantifiers. Clearly, $\mathrm{cl}^{\leq j}(\phi)$ is a closed set and $\mathrm{pd}(\mathrm{cl}^{\leq j}(\phi)) = j$.

For any closed set F, an *F-context* is any assignment of truth values to all elements of F. We say that a Kripke structure K with a single initial state s_0 *satisfies an F-context C*, written $K \models C$, if, for all $\gamma \in F$, $(K, s_0) \models \gamma$ iff $C(\gamma) = true$. An F-context is *consistent* if it is satisfied by some structure. All the F-contexts generated by our model-checking algorithm will be consistent by construction. We often identify an F-context with the elements set to true by it. For an HSM M, a node $v \in M$, an F-context C, and a formula $\gamma \in F$, we say (M, v) *satisfies* γ *in context* C, written $(M, v) \models_C \gamma$, if, for all K', $K' \models C \Rightarrow ((K(M); K'), v) \models \gamma$, where $K(M); K'$ is the Kripke structure

function MAKE_CONT(F: closed set of CTL*existential formulas,
M: HSM over $\{\gamma \in F : \mathrm{pd}(\gamma) < \mathrm{pd}(F)\}$,$C$: F-CONTEXT) **returns** HSM over F
/* We assume $M = \{M_1, \ldots, M_n\}$ with $M_i = (N_i, B_i, I_i, O_i, X_i, Y_i, C_i, R_i, E_i)$ */
[1] M_1 = TopLevelMachine(M)
[2] **for** each $\gamma \in F$ with $\gamma = E\ \rho(p(\gamma_1) \leftarrow \gamma_1, \ldots, p(\gamma_n) \leftarrow \gamma_n)$ **do**
[3] N(γ) = LTLALG($E\ \rho, M$) /* Precompute all the LTL results needed */
[4] **for** each $(\beta, \delta) \in$ SPLIT(ρ)
[5] N(β) = LTLALG(E $(\beta \wedge F\ exit), M$)
[6] **for** each $M_i \in M$ **do**
[7] Nodes$_1$(M_i,γ) = $N_i \cap$N(γ)
[8] **for** each $(\beta, \delta) \in$ SPLIT(ρ) **do**
[9] Nodes$_2$(M_i,β) = $N_i \cap$N(β)
[10] **od**
[11] **od**
[12] OLDCONT = $\emptyset$ /* Find the pairs (M_i, c) reachable from (M_1, C) */
[13] CONT = $\{(M_1, C)\}$
[14] **while** (CONT $\neq$ OLDCONT) **do**
[15] OLDCONT = CONT
[16] **for** each $(M_i, c) \in$ OLDCONT **do**
[17] **for** each $\gamma \in F$ with $\gamma =$ E$\rho(p(\gamma_1) \leftarrow \gamma_1, \ldots, p(\gamma_n) \leftarrow \gamma_n)$
[18] Sat(M_i, c, γ) =Nodes$_1$(M_i,γ) $\cup$
$\bigcup_{(\beta,\delta)\in \mathrm{SPLIT}(\gamma), \mathrm{c}(\mathrm{E}_{\delta(p(\gamma_1)\leftarrow\gamma_1,\ldots,p(\gamma_n)\leftarrow\gamma_n))=true}} \mathrm{Nodes}_2(M_i,\beta)$
[19] **for** each b $\in$ Boxes(M_i) **do**
[20] $P_{(b,i,c)} = (Y_i(b), c')$ such that $\forall \gamma \in F : c'(\gamma) = true$ iff $(b, x) \in$ Sat(M_i, c, γ)
[21] CONT = OLDCONT $\cup \{P_{(b,i,c)}\}$
[22] **od**
[23] **od**
[24] **od**
[25] /* Now build the output HSM M^*
[26] $M^* = \{M_{i,c} | (M_i, c) \in$ CONT $\}$
[27] Forall $1 \leq i \leq n$, for all $c \in$ CONT,
[28] $M_{i,c} = (N_i \times \{c\}, B_i \times \{c\}, I_i \times \{c\}, O_i \times \{c\}, X'_{i,c}, Y'_{i,c}, C'_{i,c}, R_i \times \{c\}, E_i \times \{c\})$
where
[29] $C'_{i,c} = \{((b, c), (e, c')) | (b, e) \in C_i$ and $(M_k, c') = P_{(b,i,c)}\}$
[30] For all $b \in B_i$, $Y'_{i,c}((b, c)) = (Y_i(b), c')$ with $(M_k, c') = P_{(b,i,c)}$
[31] For all $v \in N_i$, $X'_{i,c}((v, c)) = \{\gamma \in F | v \in$ Sat(M_i, c, γ)$\}$
[32] TopLevelMachine(M^*) = $M_{1,C}$
[33] **return**(M*)

Fig. 7. Construction of the context-dependent HSM.

obtained from $K(M)$ by identifying the top-level exit node of $K(M)$ with the initial state of K'.

Given a closed set F of existential formulas, an HSM M whose nodes are labeled with formulas in $\{\gamma \in F | \mathrm{pd}(\gamma) < \mathrm{pd}(F)\}$, and an F-context C, the function MAKE_CONT presented in Fig. 7 constructs a new HSM M* from multiple copies of M, each of which is indexed by an F-context c. The nodes of M* in copy (M$_j$,c) are labeled by formulas $\gamma \in F$ representing the truth value of γ in the corresponding node of M in the context c. It can be shown that any node (v, c) in M^* is labeled with $\gamma \in F$ iff $(M, v) \models_c \gamma$.

MAKE_CONT uses a variant of the LTL model-checking algorithm from Sect. 5, called LTLALG. Given a formula of the form $E\rho(p(\gamma_1,) \ldots, p(\gamma_n))$ where ρ is an LTL$^+$ formula over atomic propositions including $p(\gamma_1,) \ldots, p(\gamma_n)$, and an HSM M whose nodes are also labeled with propositions in $p(\gamma_1,) \ldots, p(\gamma_n)$, LTLALG($E\rho, M$) returns the set of nodes v of M such that $(v, \epsilon) \models E\rho$. This is done exactly as described in Sect. 5, except for the following three modifications. First, LTLALG evaluates formulas of the form $E\rho$ instead of $A\rho$. Second, we still need to define how formulas of LTL$^+$ are evaluated on M: we say that a formula

```
function CHECK(φ: existential CTL* formula, M: single-exit HSM,
        C : cl(φ)-CONTEXT) returns set of nodes in M_1
[1]  begin
[2]    M^0 = M
[3]    for{j = 0; j <pd(φ); j++}
[4]      M^{j+1}=MAKE_CONT(cl^{≤j+1}(φ), M^j, C ∩ cl^{≤j+1}(φ)))
[5]    return {v ∈ TopLevelMachine(M^{pd(φ)})|Label(v) includes φ}
     end
```

Fig. 8. CTL* model-checking algorithm.

$E\rho$ where ρ is in LTL$^+$ is satisfied in a node v of a machine M_i if there is a path w from (v, ϵ) that satisfies ρ, such that either w is infinite or w terminates at (x, ϵ), where x is the exit node of M_i. Third, we also extend the evaluation of formulas to include return nodes: we say that the return node (b, x) of a box b satisfies a formula $E\rho$ iff the corresponding exit node x satisfies $E\rho$ when b is the only element of the stack; in other words, we define $((b, x), \epsilon) \models E\rho$ iff $(x, b) \models E\rho$. It is easy to extend the LTL model-checking algorithm of Sect. 5 to meet these additional requirements.

By repeatedly invoking MAKE_CONT with $\mathrm{cl}^{\leq j}(\phi)$ for increasing values of j, $1 \leq j \leq \mathrm{pd}(\phi)$, i.e., larger and larger subsets of $\mathrm{cl}(\phi)$, one can thus evaluate CTL* formulas in a bottom-up manner. This is what is done in function CHECK presented in Fig. 8. Since any CTL* formula ϕ is a boolean combination of existential formulas ϕ_i, finding the nodes of the top-level machine M_1 of an HSM M satisfying ϕ can be reduced to finding the nodes of M_1 satisfying each ϕ_i. This is done by computing $\mathrm{CHECK}(\phi_i, M, C_\emptyset)$ where $C_\emptyset$ is the set of formulas γ in $\mathrm{cl}(\phi_i)$ that evaluate to true at a single node labeled as the exit node of M_1 and with a self-loop. Since $C_\emptyset$ is consistent, all subcontexts derived from it during the execution of the algorithm are also consistent. The correctness of the algorithm is established by the following theorem.

Theorem 8. *Given a single-exit HSM M, a node v of M_1, and an existential CTL* formula ϕ_i, (v, ϵ) satisfies ϕ_i iff v is included in the set $CHECK(\phi_i, M, C_\emptyset)$.*

An analysis of the overall complexity of CHECK reveals that the number of contexts over $F = \mathrm{cl}(\phi)$ and the number of pairs of formulas returned by SPLIT on these formulas depends only on ϕ. This implies that the size of each M^j is linear in M for any fixed ϕ. Moreover, the number of formulas on which the LTL algorithm is invoked in MAKE_CONT is bounded independently of the size of M. Hence, the run-time complexity of the function MAKE_CONT and the size of the returned HSM M^* are linear in the input HSM M for any fixed formula ϕ and closed set F. Therefore, *the CTL* model-checking problem for a single-exit HSM M can be solved in time linear in the size of M.*

7 Concluding Remarks

Function CompSummaryEdges from Sect. 5 is closely related to algorithms for solving so-called "context-free-language" reachability problems [21,17], as well as to CFL-reachability-based algorithms for such program-analysis problems

as interprocedural slicing [16] and interprocedural dataflow analysis [18,15]. In particular, the notions of path-edges and summary-edges, and the dynamic-programming technique used to compute such edges in CompSummaryEdges already appeared in this earlier work, although the cycle-detection and LTL model-checking problems considered in Sect. 5 have not been previously explored in the literature on CFL-reachability. The "transfer functions" used in [7] are also similar to the "summary-edges" used here. Results similar to those of Sect. 5 (obtained independently and contemporaneously) are reported in [2].

Thanks to Theorem 1, which provides a linear-time translation from context-free processes to single-exit HSMs, the linear-time CTL^* model-checking algorithm of Sect. 6 can also be used for CTL^* model-checking of context-free processes, and hence provides an improved upper bound for this problem: the problem can now be solved in linear-time, instead of quadratic-time.

Our other results, however, cannot even be stated in the context of context-free or pushdown processes. For example, the distinction between single-entry and multiple-entry HSMs has no obvious counterpart in the literature on pushdown automata, and the linear bounds for single-entry multiple-exit HSMs presented here could not be derived from such previous work.

References

1. A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
2. R. Alur, K. Etessami, and M. Yannakakis. Analysis of Recursive State Machines. In *To appear in Proceedings of CAV 2001*, Paris, July 2001.
3. R. Alur and R. Grosu. Modular Refinement of Hierarchic State Machines. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, pages 390–402, January 2000.
4. R. Alur, S. Kannan, and M. Yannakakis. Communicating Hierarchical State Machines. In *Proceedings of the 26th International Colloquium on Automata, Languages, and Programming*, volume 1644 of *Lecture Notes in Computer Science*, pages 169–178. Springer-Verlag, 1999.
5. R. Alur and M. Yannakakis. Model Checking of Hierarchical State Machines. In *Proceedings of the Sixth ACM Symposium on the Foundations of Software Engineering (FSE'98)*, pages 175–188, 1998.
6. A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *Proc. of CONCUR'97*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer-Verlag, 1997.
7. O. Burkart and B. Steffen. Model Checking for Context-Free Processes. In *Proc. of CONCUR'92*, 1992.
8. O. Burkart and B. Steffen. Model Checking the Full Modal Mu-Calculus for Infinite Sequential Processes. In *Proc. of ICALP'97*, volume 1256 of *Lecture Notes in Computer Science*, pages 419–429, Bologna, 1997. Springer-Verlag.
9. B. Caucal and R. Monfort. On the Transition Graphs of Automata and Grammars. In *Graph Theoretic Concepts in Computer Science*, volume 484 of *Lecture Notes in Computer Science*, pages 311–337. Springer-Verlag, 1990.
10. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier/MIT Press, Amsterdam/Cambridge, 1990.

11. E. A. Emerson and C. Lei. Efficient Model Checking in Fragments of the Propositional Mu-Calculus. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 267–278, Cambridge, June 1986.
12. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient Algorithms for Model Checking Pushdown Systems. In *Proceedings of the 12th Conference on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 232–247, Chicago, July 2000. Springer-Verlag.
13. A. Finkel, B.Willems, and P. Wolper. A Direct Symbolic Approach to Model Checking Pushdown Systems. *Electronic Notes in Theoretical Comp. Sc.*, 9, 1997.
14. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
15. S. Horwitz, T. Reps, and M. Sagiv. Demand interprocedural dataflow analysis. In *Proceedings of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 104–115, New York, NY, October 1995. ACM Press.
16. S. Horwitz, T. Reps, M. Sagiv, and G. Rosay. Speeding up slicing. In *Proceedings of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 11–20, New York, NY, December 1994. ACM Press.
17. T. Reps. Program analysis via graph reachability. *Information and Software Technology*, 40(11-12):701–726, November 1998. Special issue on program slicing.
18. T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Symp. on Princ. of Prog. Lang.*, pages 49–61, New York, NY, 1995. ACM Press.
19. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.
20. I. Walukiewicz. Pushdown Processes: Games and Model-Checking. In *Proc. 8th Conference on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 62–74, New Brunswick, August 1996. Springer-Verlag.
21. M. Yannakakis. Graph-theoretic methods in database theory. In *Proc. of the Symp. on Princ. of Database Syst.*, pages 230–242, 1990.

Symbolic Trace Analysis of Cryptographic Protocols*

Michele Boreale

Dipartimento di Sistemi e Informatica, Università di Firenze, Via Lombroso 6/17, 50134 Firenze, Italia.
boreale@dsi.unifi.it.

Abstract. A cryptographic protocol can be described as a system of concurrent processes, and analysis of the traces generated by this system can be used to verify authentication and secrecy properties of the protocol. However, this approach suffers from a state-explosion problem that causes the set of states and traces to be typically infinite or very large. In this paper, starting from a process language inspired by the spi-calculus, we propose a symbolic operational semantics that relies on unification and leads to compact models of protocols. We prove that the symbolic and the conventional semantics are in full agreement, and then give a method by which trace analysis can be carried out directly on the symbolic model. The method is proven to be complete for the considered class of properties and is amenable to automatic checking.

Keywords: spi-calculus, concurrency, formal methods for security protocols.

1 Introduction

In recent years, formal methods have proven useful in the analysis of cryptographic protocols, often revealing previously unknown attacks. A popular approach is based on the modelling of protocols as a systems of concurrent processes, described in some an appropriate language, like CSP [13,19,21] or the spi-calculus [1] – the latter an extension of the π-calculus [17]. In this setting, Abadi and Gordon advocate the use of observational equivalences to formalize and verify protocol properties [2,7]. Here, in the vein of [3,4,12,13,16,19,21], we analyze the sequences of actions (traces) that a given process may execute. As an example, a *secrecy* property like "protocol P never leaks the datum d", might be verified by adding to the description of P an 'error' action to be performed as soon as the environment learns d (the way this is done depends on the specific formalism, see e.g. [3]), and then checking that P never performs that 'error' action.

* A preliminary version of this paper has been circulated as [5]. Research partly supported by the Italian MURST Project TOSCA *(Teoria della Concorrenza, Linguaggi di Ordine Superiore e Strutture di Tipi).*

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 667–681, 2001.

The main drawback of trace analysis is that the execution of a protocol typically generates infinitely many traces. The reason lies in the modelling of the environment, whose behaviour is largely unpredictable. Rather than trying to describe this behaviour as a specific process, it is sensible to simply assume that the communication network is totally under the control of the environment. The latter can store, duplicate, hide or replace messages that travel on the network. It can also operate according to the rules followed by honest participants and synthetize new messages by pairing, decryption, encryption and creation of fresh nonces and keys, or by arbitrary combinations of these operations (this approach seems to date back to Dolev and Yao [11]). Thus, an agent waiting for an input at a given moment may expect any of the infinitely many messages the environment can produce and send over the network. This leads to a state explosion that makes the protocol model, typically a state-transition graph, infinite (more precisely, infinite-branching). In practice, those approaches that rely on *model checking* [13,19,21] cut down the model to a convenient finite size by imposing upper-bounds to the critical parameters (number of keys, number of pairing and encryption in messages,...). Exhaustive exploration of the state-space is then possible by standard techniques. However, this approach makes correctness in the general case (completeness) very difficult to establish – though some progress has recently been made [15,20]. Furthermore, even when those upper-bounds can be justified and the model is finite, the branching factor of input actions may cause the number of states and traces to explode as larger systems are considered.

In this paper we explore an alternative approach to trace analysis of cryptographic protocols. As a base language, we consider a variant of the spi-calculus, but this choice is not critical for the development of the theory. The idea is to replace the infinitely many transitions arising from an input action by a single *symbolic* transition, and to represent the received message as a variable. Constraints on this variable are accumulated as the execution proceeds. Let us see this in more detail. In the variant of the spi-calculus we use, the receiver of a message is written as $\mathsf{a}(x).R$, where a is an arbitrary label, x is the input variable and R is the continuation. The conventional ('concrete') operational semantics of the language requires x to be instantiated with each message that can possibly be received from the network, and this causes the state explosion. In our symbolic semantics, x is not instantiated immediately, rather constraints on its value are added as needed. These constraints take the form of *most general unifiers*. As an example, suppose that a process P, after receiving a message x, tries decryption of x using key k, and, if this succeeds, calls y the result and proceeds like P'. This is written as $P \stackrel{\text{def}}{=} \mathsf{a}(x).\,\mathsf{case}\,x\,\mathsf{of}\,\{y\}_k\,\mathsf{in}\,P'$. We represent a state of the protocol as a pair (σ, Q), where σ is the trace of process' past actions and Q is a process term. The two initial symbolic transitions of $(\epsilon,\ P)$ will be:

$$(\epsilon,\ P) \ \longrightarrow_{\mathsf{s}}\ (\mathsf{a}\langle x\rangle,\ \mathsf{case}\,x\,\mathsf{of}\,\{y\}_k\,\mathsf{in}\,P') \ \longrightarrow_{\mathsf{s}}\ (\mathsf{a}\langle\{y\}_k\rangle,\ P'[\{y\}_k/x])$$

where $[\{y\}_k/x]$ is the most general unifier for x and $\{y\}_k$. In general, protocols not using replication/recursion will generate finitely many symbolic transitions. The resulting model is rather compact: sequential processes will generate just one maximal symbolic trace, while, for parallel compositions, traces will be obtained

by the usual interleaving. We prove that the symbolic and the conventional semantics are in full agreement, and then give a method by which trace analysis can be carried out directly on the symbolic traces. We focus our attention on a specific class of properties, those of the form "in every execution of the protocol, action α happens prior to action β", for given α and β. As we shall see, this scheme is flexible enough to express interesting forms of authentication and secrecy. The method is proven to be complete for the language we consider, and is easily mechanizable; this immediately yields decidability of trace analysis for the considered language. When a protocol does not satisfy a property, the method also gives an easy way to compute an attack, i.e. a trace that violates the property. A prototype implementation of the method is already available [6].

The language we consider does not contain replication/recursion operators, that would make trace analysis undecidable (see e.g. [12, 10]). Thus we consider only protocols with a bounded number of participants. However, recent work by Lowe [15] and Stoller [22] indicates that, in some cases, it is possible to reduce the analysis of an unbounded protocol to the analysis of a protocol with a bounded, statically determined number of participants.

The symbolic approach to the analysis of security protocols has been explored by other authors, including Amadio and Lugiez [4] and Huima [12]. Discussion with these and other related work can be found in the concluding section.

The rest of the paper is organized a follows. The language, a variant of the spi-calculus with shared-key cryptography, is introduced in Section 2, along with its conventional operational semantics. Section 3 introduces trace analysis. The symbolic operational semantics, and its agreement with the conventional semantics, are discussed in Section 4. Section 5 presents the symbolic method for trace analysis, at the core of which is the concept of *refinement*. For the sake of presentation, the language of Section 2 does not include the restriction operator, whose treatment is postponed to Section 6. Section 7 contains discussion on related work and a few concluding remarks.

2 The Language

Syntax. (Table 1) We presuppose three countable disjoint sets: $\mathcal{L}$, $\mathcal{N}$ and $\mathcal{V}$. The set $\mathcal{L}$ of *labels* is ranged over by $\mathsf{a}, \mathsf{b}, \ldots$. The set of $\mathcal{N}$ of *names* is partitioned into two countable sets, a set $\mathcal{LN}$ of *local names* $a, b, \ldots$ and a set $\mathcal{EN}$ of *environmental names* $\underline{a}, \underline{b}, \ldots$: these sets represent the basic data (keys, nonces,...) initially known to the process and to the environment, respectively. The set $\mathcal{V}$ of *variables* is ranged over by $x, y, \ldots$. The set $\mathcal{N} \cup \mathcal{V}$ is ranged over by letters $u, v, \ldots$. Names and variables can be used to build compound *messages*, in $\mathcal{M}$, via shared-key encryption and pairing. In particular, $\{M\}_k$ represents the message obtained by encrypting M (the *argument*) under name k (the *key*), using a shared-key encryption system. We allow pairing and encryptions to be arbitrarily nested, but only permit atomic keys. *Terms* are obtained by closing messages under substitutions (they are just 'garbage' that can be generated at run-time, with no semantical significance).

The syntax of *agent expressions*, in $\mathcal{A}$, is taken essentially from the spi-calculus [1]. Agent $\mathsf{case}\,\{M\}_h\,\mathsf{of}\,\{y\}_k\,\mathsf{in}\,A$ tries decryption of $\{M\}_h$ using k as a key: if this is possible (that is, if $k = h$), the result of the decryption, M, is bound to y, and the agent proceeds like A, otherwise, the whole expression is stuck. Similarly, $\mathsf{pair}\,M\,\mathsf{of}\,\langle x, y\rangle\,\mathsf{in}\,A$ tries to split M into two components and call them x and y. A difference from spi/π-calculus is that, in our case, input and output labels ($\mathsf{a}, \mathsf{b}, \ldots$) must not be regarded as a channels (as already noted, we assume just one public network), but rather as 'tags' attached to process actions for ease of reference. Also, the only useful case for output is when ζ is a message, otherwise the whole agent is stuck.

Given the presence of binders for variables, notions of *free variables*, $\mathrm{v}(A) \subseteq \mathcal{V}$ and *alpha-equivalence* arise as expected. We shall identify alpha-equivalent agent expressions. For any M and u, $[^M/_u]$ denotes the operation of substituting the free occurrences of u by M. An agent expression A is said to be *closed* or a *process* if $\mathrm{v}(A) = \emptyset$; the set of processes $\mathcal{P}$ is ranged over by $P, Q, \ldots$. Local names and environmental names occurring in A are denoted by $\mathrm{ln}(A)$ and $\mathrm{en}(A)$, respectively. A process P is *initial* if $\mathrm{en}(P) = \emptyset$. These notations are extended to terms, messages, and tuples/string/sets of such objects, component-wise. We shall also use such abbreviations as $\mathrm{ln}(M, P, Q)$ to mean $\mathrm{ln}(M) \cup \mathrm{ln}(P) \cup \mathrm{ln}(Q)$.

Table 1. Syntax

$m, n, \ldots$	names $\mathcal{N}$	$x, y, \ldots$ variables $\mathcal{V}$
$a, b, \ldots, h, k, \ldots$	local names $\mathcal{LN}$	
$\underline{a}, \underline{b}, \ldots, \underline{h}, \underline{k}, \ldots$	environmental names $\mathcal{EN}$	
$u, v, \ldots$	variables or names $\mathcal{V} \cup \mathcal{N}$	$\mathsf{a}, \mathsf{b}, \ldots$ labels $\mathcal{L}$

$M, N ::= u \mid \{M\}_u \mid \langle M, N\rangle$	messages $\mathcal{M}$	
$\eta, \zeta ::= u \mid \{\zeta\}_\eta \mid \langle \zeta, \eta\rangle$	terms $\mathcal{Z}$	
$A, B ::=$	agents $\mathcal{A}$	
$\mathbf{0}$	(null)	
$\mid \mathsf{a}(x).\, A$	(input)	
$\mid \overline{\mathsf{a}}\langle\zeta\rangle.\, A$	(output)	
$\mid \mathsf{case}\,\zeta\,\mathsf{of}\,\{y\}_\eta\,\mathsf{in}\,A$	(decryption)	
$\mid \mathsf{pair}\,\zeta\,\mathsf{of}\,\langle x, y\rangle\,\mathsf{in}\,A$	(selection)	
$\mid [\zeta = \eta]A$	(matching)	
$\mid A \parallel B$	(parallel composition)	

The occurrences of variables x and y in (input), (decryption) and (selection) are bound.

Example 1 (the wide-mouthed frog protocol, WMF [8]). Principals A and B share two secret keys, k_{AS} and k_{BS} respectively, with a server S. The purpose of the protocol is that of establishing a new secret key k between A and B, which A may

use to send a confidential datum d to B. For the sake of simplicity, we suppose that the protocol is always started by A. The protocol and its translation in spi-calculus, WMF, are described below:

$$\begin{array}{ll}
1.\ A \longrightarrow S : \{k\}_{k_{AS}} & A \stackrel{\text{def}}{=} \overline{\mathsf{a1}}\langle\{k\}_{k_{AS}}\rangle.\,\overline{\mathsf{a2}}\langle\{d\}_k\rangle.\,A' \\
2.\ S \longrightarrow B : \{k\}_{k_{BS}} & S \stackrel{\text{def}}{=} \mathsf{s1}(x).\,\mathsf{case}\ x\ \mathsf{of}\ \{x'\}_{k_{AS}}\ \mathsf{in}\ \overline{\mathsf{s2}}\langle\{x'\}_{k_{BS}}\rangle.\,\mathbf{0} \\
3.\ A \longrightarrow B : \{d\}_k. & B \stackrel{\text{def}}{=} \mathsf{b1}(y).\,\mathsf{case}\ y\ \mathsf{of}\ \{y'\}_{k_{BS}}\ \mathsf{in}\ \mathsf{b2}(z).\,\mathsf{case}\ z\ \mathsf{of}\ \{z'\}_{y'}\ \mathsf{in}\ B' \\
 & WMF \stackrel{\text{def}}{=} A \parallel S \parallel B\,.
\end{array}$$

Agents A' and B' represent the behaviour of A and B, respectively, after the protocol has been completed. The above description just accounts for a single instance of the protocol. The case of $n > 1$ instances is described by composing n copies of A, S and B in parallel and then appropriately renaming the instance-dependent quantities (k, d, n_A and n_B above).

Operational semantics. The basic semantics of the calculus is given in terms of a transition relation $\longrightarrow$, which we will sometimes refer to as 'concrete' (as opposed to the 'symbolic' one we shall introduce later on). We will find it convenient to model a state of the system as a pair (s, P), where, s records the current environment's knowledge (i.e. the sequence of messages the environment has "seen" travelling on the network up to that moment) and P is a process. Similarly to [3,4,9], we characterize the messages that the environment can *produce* (or deduce) at a given moment, starting from the current knowledge s, via a deductive system. These concepts are formalized below.

Definition 1 (the deductive system). Let $S \subseteq_{\text{fin}} \mathcal{M}$ and M a message. We let $\vdash$ be the least binary relation generated by the deductive system in Table 2. If $S \vdash M$ we say that S *can produce* M. ◇

Table 2. Deductive system ($\vdash$)

$$\text{(Ax)}\ \frac{}{S \vdash M}\ M \in S \qquad \text{(Env)}\ \frac{}{S \vdash \underline{a}}\ \underline{a} \in \mathcal{EN}$$

$$\text{(Proj}_1\text{)}\ \frac{S \vdash \langle M, N\rangle}{S \vdash M} \qquad \text{(Proj}_2\text{)}\ \frac{S \vdash \langle M, N\rangle}{S \vdash N} \qquad \text{(Pair)}\ \frac{S \vdash M \quad S \vdash N}{S \vdash \langle M, N\rangle}$$

$$\text{(Dec)}\ \frac{S \vdash \{M\}_u \quad S \vdash u}{S \vdash M} \qquad \text{(Enc)}\ \frac{S \vdash M \quad S \vdash u}{S \vdash \{M\}_u}$$

Note that, whatever S, the set of messages that S can produce is infinite, due to rules Env, Pair and Enc. An *action* is a term of the form $\mathsf{a}\langle M\rangle$ (*input* action)

or $\overline{\mathsf{a}}\langle M\rangle$ (*output* action), for a a label and M a message. The set of actions Act is ranged over by $\alpha, \beta, \ldots$, while the set Act^* of strings of actions is ranged over by $s, s', \ldots$. String concatenation is written '$\cdot$'. We denote by $\mathrm{act}(s)$ and $\mathrm{msg}(s)$ the set of actions and messages, respectively, appearing in s. A string s is *closed* if $\mathrm{v}(s) = \emptyset$ (note that s does not contain binders) and *initial* if $\mathrm{en}(s) = \emptyset$. In what follows, we write $s \vdash M$ for $\mathrm{msg}(s) \vdash M$. We are now set to define *traces*, that is sequences of actions that may result from the interaction between a process and its environment. In traces, each message received by the process (input message) must be deducible from the knowledge the environment has previously acquired. In *configurations*, this knowledge is explicitly recorded.

Definition 2 (traces and configurations). A *trace* is a closed string $s \in Act^*$ such that for each s_1, s_2 and $\mathsf{a}\langle M\rangle$, if $s = s_1 \cdot \mathsf{a}\langle M\rangle \cdot s_2$ then $s_1 \vdash M$.

A *configuration*, written as $\langle s,\ P\rangle$, is a pair consisting of a trace s and a process P. A configuration is *initial* if $\mathrm{en}(s, P) = \emptyset$. Configurations are ranged over by $\mathcal{C}, \mathcal{C}', \ldots$. ◇

Table 3. Transition relation ($\longrightarrow$).

$$\textsc{(Inp)}\ \langle s,\ \mathsf{a}(x).P\rangle \longrightarrow \langle s \cdot \mathsf{a}\langle M\rangle,\ P[M/x]\rangle \qquad s \vdash M,\ M \text{ closed}$$

$$\textsc{(Out)}\ \langle s,\ \overline{\mathsf{a}}\langle M\rangle.P\rangle \longrightarrow \langle s \cdot \overline{\mathsf{a}}\langle M\rangle,\ P\rangle$$

$$\textsc{(Case)}\ \langle s,\ \mathsf{case}\ \{\zeta\}_\eta\ \mathsf{of}\ \{y\}_\eta\ \mathsf{in}\ P\rangle \longrightarrow \langle s,\ P[\zeta/y]\rangle$$

$$\textsc{(Select)}\ \langle s,\ \mathsf{pair}\ \langle\zeta,\eta\rangle\ \mathsf{of}\ \langle x,y\rangle\ \mathsf{in}\ P\rangle \longrightarrow \langle s,\ P[\zeta/x,\ \eta/y]\rangle$$

$$\textsc{(Match)}\ \langle s,\ [\zeta = \zeta]P\rangle \longrightarrow \langle s,\ P\rangle$$

$$\textsc{(Par)}\ \frac{\langle s,\ P\rangle \longrightarrow \langle s',\ P'\rangle}{\langle s,\ P \parallel Q\rangle \longrightarrow \langle s',\ P' \parallel Q\rangle}$$

plus symmetric version of (Par).

The concrete transition relation on configurations is defined by the rules in Table 3. Rule (Inp) makes the transition relation infinite-branching, as M ranges over the infinite set $\{M : s \vdash M,\ M \text{ closed}\}$. Note that, for the sake of uniformity, in rule (Case) we allow decryption of arbitrary terms (which use a possibly compound key η), despite the fact that we only allow sending and receiving of messages. Similarly, we allow comparison between arbitrary terms in (Match). Finally, no handshake communication is provided (rule (Par)): all messages go through the environment.

3 Trace Analysis

Given a configuration $\langle s,\ P\rangle$ and a trace s', we say that $\langle s,\ P\rangle$ *generates* s', written $\langle s,\ P\rangle \searrow s'$, if $\langle s,\ P\rangle \longrightarrow^* \langle s',\ P'\rangle$ for some P'. Given a string of actions $s \in Act^*$, and actions α and β, we say that α *occurs prior to* β *in* s if whenever $s = s'\cdot\beta\cdot s''$ then $\alpha \in \mathrm{act}(s')$. We let ρ range over ground substitutions, i.e. finite maps from a set $\mathrm{dom}(\rho) \subseteq \mathcal{V}$ to closed messages; $t\rho$ denotes the result of replacing each $x \in \mathrm{v}(t) \cap \mathrm{dom}(\rho)$ by $\rho(x)$. The properties of configurations we are interested in are defined below.

Definition 3 (properties). Let α and β be actions, with $\mathrm{v}(\alpha) \subseteq \mathrm{v}(\beta)$, and let s be a trace. We write $s \models \alpha \hookleftarrow \beta$, or s *satisfies* $\alpha \hookleftarrow \beta$, if for each ground substitution ρ it holds that $\alpha\rho$ occurs prior to $\beta\rho$ in s. We say that a configuration $\mathcal{C}$ *satisfies* $\alpha \hookleftarrow \beta$, and write $\mathcal{C} \models \alpha \hookleftarrow \beta$, if all traces generated by $\mathcal{C}$ satisfy $\alpha \hookleftarrow \beta$. $\diamond$

Note that the variables in α and β can be thought of as being universally quantified (so any consistent renaming of these variables does not change the set of traces and configurations that satisfy $\alpha \hookleftarrow \beta$). In practice, the scheme $\alpha \hookleftarrow \beta$ permits formalizing all forms of authentication in Lowe's hierarchy [14], except the most demanding one (but can be easily modified to include this one as well). As we shall see, the scheme also permits expressing secrecy as a reachability property, in the style of [3]. To this purpose, it is convenient to assume a fixed 'absurd' action $\bot$ that is nowhere used in agent expressions. Thus the formula $\bot \hookleftarrow \alpha$ expresses that no instance of action α should ever take place, and can be used to encode reachability.

Example 2 (authentication and secrecy in WMF). We discuss the use of properties on the simple protocol WMF (Example 1), but our considerations are indeed quite general.

A property of WMF that one would like to check is the following: if B accepts as 'good' a datum d encrypted under key k (step 3), then this message has actually been sent by A. This is a form of *authentication*. In order to formalize this particular property, we make B explicitly declare if, upon completion of the protocol, a particular message has been accepted. That is, we consider the process $B_{auth} \stackrel{\text{def}}{=} \mathsf{b1}(y).\,\mathsf{case}\ y\ \mathsf{of}\ \{y'\}_{k_{BS}}\ \mathsf{in}\ \mathsf{b2}(z).\,\mathsf{case}\ z\ \mathsf{of}\ \{z'\}_{y'}\ \mathsf{in}\ (B' \parallel \overline{\mathsf{accept}}\langle z\rangle.\,\mathbf{0})$ instead of B, and $WMF_{auth} \stackrel{\text{def}}{=} A \parallel S \parallel B_{auth}$ instead of WMF. We have to check that, in every trace of WMF_{auth}, every $\overline{\mathsf{accept}}$ is preceded by the corresponding $\overline{\mathsf{a2}}$. More formally, we have to check that $\langle \epsilon,\ WMF_{auth}\rangle \models \overline{\mathsf{a2}}\langle t\rangle \hookleftarrow \overline{\mathsf{accept}}\langle t\rangle$ (where t is any variable).

Another important property is *secrecy*: the environment should never learn the confidential datum d. Following [3], we can formalize this property by considering a version of WMF that also includes a 'guardian' listening to the public network: $WMF_{secr} \stackrel{\text{def}}{=} WMF \parallel \mathsf{guard}(x).\,\mathbf{0}$. Evidently, WMF generates a trace s s.t. $s \vdash d$ (i.e. the environment may learn d) if and only if $\langle \epsilon,\ WMF_{secr}\rangle$ generates a trace containing action $\mathsf{guard}\langle d\rangle$. Thus, we have to check that action $\mathsf{guard}\langle d\rangle$ never takes place, i.e. that $\langle \epsilon,\ WMF_{secr}\rangle \models \bot \hookleftarrow \mathsf{guard}\langle d\rangle$.

4 Symbolic Semantics

'Concrete' traces and configurations can be given a symbolic counterpart, which may contain free variables.

Definition 4 (symbolic traces and configurations). A *symbolic trace* is string $\sigma \in Act^*$ such that: (a) $\mathrm{en}(\sigma) = \emptyset$, and (b) for each σ_1, σ_2, α and x, if $\sigma = \sigma_1 \cdot \alpha \cdot \sigma_2$ and $x \in \mathrm{v}(\alpha) - \mathrm{v}(\sigma_1)$ then α is an input action. Symbolic traces are ranged over by $\sigma, \sigma', \ldots$.

A *symbolic configuration*, written $\langle \sigma,\ A \rangle_{\mathrm{s}}$, is a pair composed by a symbolic trace σ and an agent A, such that $\mathrm{en}(A) = \emptyset$ and $\mathrm{v}(A) \subseteq \mathrm{v}(\sigma)$. ◇

Note that, due to condition (b) in the definition, e.g. $\overline{\mathsf{a}}\langle x \rangle \cdot \mathsf{a}\langle \{h\}_x \rangle$ is not a symbolic trace, while $\mathsf{a}\langle \{h\}_x \rangle \cdot \overline{\mathsf{a}}\langle x \rangle$ is. Let us now recall some standard terminology about substitutions. A substitution θ is a finite partial map from $\mathcal{V}$ to $\mathcal{M}$ and, for any object (i.e. variable, message, process, trace,...) t, we denote by $t\theta$ the result of applying θ to t. A substitution θ is a *unifier of* t_1 *and* t_2 if $t_1\theta = t_2\theta$. We denote by $\mathrm{mgu}(t_1, t_2)$ a chosen *most general unifier (mgu)* of t_1 and t_2, that is a unifier θ such that any other unifier can be written as a composition of substitutions $\theta\theta'$ for some θ'.

The transition relation on symbolic configurations, $\longrightarrow_{\mathrm{s}}$, is defined by the rules in Table 4. There, a function $\mathrm{new}_{\mathcal{V}}(\cdot)$ is assumed such that, for any given $V \subseteq_{\mathrm{fin}} \mathcal{V}$, $\mathrm{new}_{\mathcal{V}}(V)$ is a variable not in V. Note that, differently from the concrete semantics, input variables are not instantiated immediately, in the input rule ($\textsc{Inp}_{\mathrm{s}}$). Rather, constraints on these variables are added as soon as they are needed, and recorded via mgu's. This may occur due to rules ($\textsc{Case}_{\mathrm{s}}$), ($\textsc{Select}_{\mathrm{s}}$) and ($\textsc{Match}_{\mathrm{s}}$). In the example below, after the first input step, variable x gets instantiated to name b due to a ($\textsc{Match}_{\mathrm{s}}$)-reduction:

$$\langle \epsilon,\ \mathsf{a}(x).\,[x = b]P \rangle_{\mathrm{s}} \ \longrightarrow_{\mathrm{s}}\ \langle \mathsf{a}\langle x \rangle,\ [x = b]P \rangle_{\mathrm{s}} \ \longrightarrow_{\mathrm{s}}\ \langle \mathsf{a}\langle b \rangle,\ P[{}^{b}\!/_{x}] \rangle_{\mathrm{s}}\ .$$

The side condition on B' in ($\textsc{Par}_{\mathrm{s}}$) ensures that constraints are propagated across parallel components sharing variables, like in the following ($\textsc{Match}_{\mathrm{s}}$)-reduction: $\langle \sigma,\ [x = M]A \parallel \overline{\mathsf{a}}\langle x \rangle.\, B) \rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}} \langle \sigma[{}^{M}\!/_{x}],\ A[{}^{M}\!/_{x}] \parallel \overline{\mathsf{a}}\langle M \rangle.\, B[{}^{M}\!/_{x}] \rangle_{\mathrm{s}}$.

Whenever $\langle \sigma,\ A \rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}}^{*} \langle \sigma',\ A' \rangle_{\mathrm{s}}$ for some A', we say that $\langle \sigma,\ A \rangle_{\mathrm{s}}$ *symbolically generates* σ', and write $\langle \sigma,\ A \rangle_{\mathrm{s}} \searrow_{\mathrm{s}} \sigma'$. The relation $\longrightarrow_{\mathrm{s}}$ is finite-branching. This implies that each configuration generates a finite number of symbolic traces. It is important to stress that many symbolic traces are in fact 'garbage' – jumbled sequences of actions that cannot be instantiated to give a concrete trace. This is the case, e.g., for the trace $\mathsf{a}\langle \{x\}_k \rangle \cdot \overline{\mathsf{a}}\langle x \rangle$, which is symbolically generated by $\langle \epsilon,\ P \rangle_{\mathrm{s}}$, where $P \stackrel{\mathrm{def}}{=} \mathsf{a}(y).\, \mathsf{case}\ y\ \mathsf{of}\ \{x\}_k\ \mathsf{in}\ \overline{\mathsf{a}}\langle x \rangle.\, \mathbf{0}$. To state soundness and completeness of $\longrightarrow_{\mathrm{s}}$ w.r.t. $\longrightarrow$, we need a notion of consistency for symbolic traces, given below.

Definition 5 (solutions of symbolic traces). Given a symbolic trace σ and a ground substitution ρ, we say that ρ *satisfies* σ if $\sigma\rho$ is a trace. In this case, we also say that $\sigma\rho$ is a *solution* of σ, and that σ is *consistent*. ◇

Table 4. Symbolic transition relation ($\longrightarrow_{\mathrm{s}}$)

$$(\textsc{Inp}_{\mathrm{s}})\ \langle \sigma,\ \mathsf{a}(x).\,A\rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}} \langle \sigma \cdot \mathsf{a}\langle x\rangle,\ A\rangle_{\mathrm{s}}$$

$$(\textsc{Out}_{\mathrm{s}})\ \langle \sigma,\ \overline{\mathsf{a}}\langle M\rangle.\,A\rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}} \langle \sigma \cdot \overline{\mathsf{a}}\langle M\rangle,\ A\rangle_{\mathrm{s}}$$

$$(\textsc{Case}_{\mathrm{s}})\ \langle \sigma,\ \mathsf{case}\,\zeta\,\mathsf{of}\,\{x\}_{\eta}\,\mathsf{in}\,A\rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}} \langle \sigma\theta,\ A\theta\rangle_{\mathrm{s}} \quad \theta = \mathrm{mgu}(\zeta, \{x\}_{\eta})$$

$$(\textsc{Select}_{\mathrm{s}})\ \langle \sigma,\ \mathsf{pair}\,\zeta\,\mathsf{of}\,\langle x, y\rangle\,\mathsf{in}\,A\rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}} \langle \sigma\theta,\ A\theta\rangle_{\mathrm{s}} \quad \theta = \mathrm{mgu}(\zeta, \langle x, y\rangle)$$

$$(\textsc{Match}_{\mathrm{s}})\ \langle \sigma,\ [\zeta = \eta]A\rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}} \langle \sigma\theta,\ A\theta\rangle_{\mathrm{s}} \quad \theta = \mathrm{mgu}(\zeta, \eta)$$

$$(\textsc{Par}_{\mathrm{s}})\ \frac{\langle \sigma,\ A\rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}} \langle \sigma',\ A'\rangle_{\mathrm{s}}}{\langle \sigma,\ A \parallel B\rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}} \langle \sigma',\ A' \parallel B\theta\rangle_{\mathrm{s}}} \quad \text{where } \sigma' = \sigma\theta, \text{ for some } \theta$$

plus symmetric version of ($\textsc{Par}_{\mathrm{s}}$). In the above rules, it is assumed that:
(i) $x = \mathrm{new}_{\mathcal{V}}(V)$ – where V is the set of free variables in the source configuration
(ii) $y = \mathrm{new}_{\mathcal{V}}(V \cup \{x\})$, and
(iii) $\mathrm{msg}(\sigma)\theta \subseteq \mathcal{M}$.

Theorem 6 (soundness and completeness). *Let $\mathcal{C}$ be an initial configuration and s a trace. Then $\mathcal{C} \searrow s$ if and only if there is σ s.t. $\mathcal{C} \searrow_{\mathrm{s}} \sigma$ and s is a solution of σ.*

Proof: By transition induction on $\longrightarrow$ and $\longrightarrow_{\mathrm{s}}$, and then by induction on the length of traces. □

Any given configuration generates only finitely many symbolic traces. Thus, by the previous theorem, the task of checking $\mathcal{C} \models \alpha \hookleftarrow \beta$ is reduced to analysing each of these symbolic traces in turn. To do this, we need at least a method to tell whether any given symbolic trace is consistent or not. More precisely, the previous theorem reduces the problem $\mathcal{C} \models \alpha \hookleftarrow \beta$ to the following, that will be faced in the next section.

Symbolic trace analysis problem (STAP) 5. Given two actions α, β ($\mathrm{v}(\alpha) \subseteq \mathrm{v}(\beta)$) and a symbolic trace σ, check whether or not each solution s of σ satisfies $\alpha \hookleftarrow \beta$.

We write $\sigma \models \alpha \hookleftarrow \beta$ if the answer to STAP with σ, α and β is 'yes'. Note that STAP is a non-trivial problem: one has to consider *every* solution of σ, and there may be infinitely many of them.

5 Refinement

As a first step towards devising a method for STAP, let us consider the simpler problem of checking consistency of a symbolic trace. Existence of solutions (i.e.

ground instances that are traces) of a symbolic trace σ depends on the form of input actions in σ. For example, the symbolic trace $\sigma_0 = \overline{\mathsf{a}}\langle h\rangle \cdot \mathsf{b}\langle\{x\}_k\rangle$ ($h \neq k$) has no solution, because no matter which m is substituted for x, we have $\{h\} \not\vdash \{m\}_k$. On the contrary, in $\sigma_1 = \overline{\mathsf{c}}\langle k\rangle \cdot \mathsf{c}\langle\{x\}_k\rangle$, instantiating x with any environmental name $\underline{a} \in \mathcal{EN}$ will give a solution, because $\{k\} \vdash \{\underline{a}\}_k$. Yet a different case is $\sigma_2 = \overline{\mathsf{c}}\langle\{a\}_k\rangle \cdot \overline{\mathsf{c}}\langle\{b\}_k\rangle \cdot \mathsf{c}\langle\{x\}_k\rangle$. Since $\{\{a\}_k, \{b\}_k\} \not\vdash k$, there are only two ways of getting a solution of σ_2: to unify $\{x\}_k$ with either $\{a\}_k$ or $\{b\}_k$. These examples suggest that it should be possible to check consistency of a symbolic trace by gradually instantiating its variables until a trace is obtained. We shall call this process *refinement*. In order to formalize this concept, we first need to lift the definition of 'trace' to the non-ground case. This requires a few more notations and concepts.

In refinement, we shall consider both ordinary variables and *marked* variables; roughly, the latters can only be instantiated to messages that the environment can produce. This is made precise in the sequel. We consider a new set $\widehat{\mathcal{V}}$ of marked variables, which is in bijection with $\mathcal{V}$ via a mapping $\hat{\cdot}$: thus variables $x, y, z, \ldots$ have marked versions $\hat{x}, \hat{y}, \hat{z}, \ldots$. Marked messages are messages that may also contain marked variables, and marked symbolic traces are defined similarly. The deduction relation $\vdash$ is extended to marked messages by adding the new axiom

$$(\textsc{Mvar})\ \frac{}{S \vdash \hat{x}} \quad \hat{x} \in \widehat{\mathcal{V}}$$

to the system of Table 2. For any $\hat{x}$ and any sequence σ, we denote by $\sigma\backslash\hat{x}$ the longest prefix of σ not containing $\hat{x}$. The satisfaction relation is extended to marked symbolic traces as follows:

Definition 7. Let σ be a marked symbolic trace and ρ be a ground substitution. We say that ρ *satisfies* σ if $\sigma\rho$ is a trace and, for each $\hat{x} \in \mathrm{v}(\sigma)$, it holds that $(\sigma\backslash\hat{x})\rho \vdash \rho(\hat{x})$. We also say that $\sigma\rho$ is a *solution* of σ, and that σ is *consistent*. ◇

The terminology introduced above agrees with Definition 5 when σ does not contain marked variables. We can give now the definition of *solved form*, that lifts the definition of trace to the non-ground case (note that this definition is formally the same as Def. 2)

Definition 8 (solved forms). Let σ be a marked symbolic trace. We say σ is in *solved form (sf)* if for every σ_1, $\mathsf{a}\langle M\rangle$ and σ_2 s.t. $\sigma = \sigma_1 \cdot \mathsf{a}\langle M\rangle \cdot \sigma_2$ it holds that $\sigma_1 \vdash M$. ◇

Solved forms are consistent: the next lemma gives us a specific way to instantiate a solved form so as to get a trace.

Lemma 9. *Let σ be in solved form and let ρ be any substitution from $\mathrm{v}(\sigma)$ to $\mathcal{EN}$. Then ρ satisfies σ.*

A key concept of refinement is that of decomposing a message into its irreducible components, those that cannot be further split or decrypted using the knowledge of a given σ.

Definition 10 (decomposition of messages). Let σ be a marked symbolic trace. We define the sets

- $\mathbf{I}(\sigma) \stackrel{\text{def}}{=} \{M |\, \sigma \vdash M$ and either$M \in \mathcal{LN} \cup \mathcal{V}$ or $M = \{N\}_u$ for some u s.t. $\sigma \nvdash u\}$.
- $[M]_\sigma$ by induction on M as follows:

$$[u]_\sigma = \{u\} - (\widehat{\mathcal{V}} \cup \mathcal{EN})$$
$$[\langle M, N\rangle]_\sigma = [M]_\sigma \cup [N]_\sigma$$
$$[\{M\}_u]_\sigma = \begin{cases} \{\{M\}_u\} & \text{if } \sigma \nvdash u \\ [M]_\sigma & \text{if } \sigma \vdash u \end{cases}$$

◇

The irreducible components of σ, $\mathbf{I}(\sigma)$, are the building blocks of messages that can be produced by σ. This is the content of the next proposition, that makes the relationship among $\mathbf{I}(\sigma)$, $[M]_\sigma$ and $\vdash$ precise.

Proposition 11. *Let σ be a marked symbolic trace. Then $\sigma \vdash M$ if and only if $[M]_\sigma \subseteq \mathbf{I}(\sigma)$.*

There are two points worth noting with respect to the above proposition. First, $\mathbf{I}(\sigma)$ is finite and can be easily computed by an iterative procedure, thus the proposition gives us an effective method to decide $\sigma \vdash M$; this also implies that the set of solved forms is decidable. Second, the proposition suggests a strategy for refining a generic σ to a solved form: for any input message M in σ, one tries to make the condition $[M]_{\sigma'} \subseteq \mathbf{I}(\sigma')$ true, for the appropriate prefix σ' of σ. We are now ready to define refinement formally. In the sequel, we shall use the following notations. We write '$t \in_\theta S$' for: there is $t' \in S$ s.t. $\theta = \text{mgu}(t, t')$; when $\widetilde{y}$ is a set of variables, we denote by $[\widetilde{y}/\widehat{\widetilde{y}}]$ the substitution that for each $x \in \widetilde{y}$ maps $\hat{x}$ to x.

Table 5. Refinement ($\succ$)

Let σ be a marked symbolic trace, and assume $\sigma = \sigma' \cdot \mathsf{a}\langle M\rangle \cdot \sigma''$, where σ' is the longest prefix of σ that is in solved form. Assume $N \in [M]_{\sigma'} - \mathbf{I}(\sigma')$.

$$(\text{Ref}_1)\ \frac{N \notin \mathcal{V},\ N \in_\theta \mathbf{I}(\sigma'),\ \widetilde{y} = \{x |\, \hat{x} \in \text{v}(\sigma) \text{ and } (\sigma\theta)\backslash\hat{x} \text{ is shorter than } \sigma\backslash\hat{x}\}}{\sigma \ \succ\ \sigma\theta[\widetilde{y}/\widehat{\widetilde{y}}]}$$

$$(\text{Ref}_2)\ \frac{N = x \text{ or } N = \{N'\}_x}{\sigma \ \succ\ \sigma[\hat{x}/x]}$$

Definition 12 (refinement). We let *refinement*, written $\succ$, be the least binary relation over marked symbolic traces generated by the two rules in Table 5.

◇

Rule (REF_1) implements the basic step: an element N in the decomposition of M gets instantiated, via θ, to an irreducible component of some past message (to be found in $\mathbf{I}(\sigma')$). E.g., consider again the above $\sigma_2 = \overline{\mathsf{c}}\langle\{a\}_k\rangle \cdot \overline{\mathsf{c}}\langle\{b\}_k\rangle \cdot \mathsf{c}\langle\{x\}_k\rangle$: its possible refinements are $\sigma_2 \succ \sigma_2[^a/_x]$ and $\sigma_2 \succ \sigma_2[^b/_x]$, and the refined traces are in sf. By rule (REF_2), one may choose to mark a variable x, that will be considered as part of the environment's knowledge in subsequent refinement. Sometimes marked variables need to be 'unmarked' back to variables, and this is achieved via the renaming $[\widetilde{y}/\widetilde{\hat{y}}]$ in (REF_1).[1]

Refinement is repeatdly applied until some solved form is reached. It is important to realize that the reflexive and transitive closure $(\succ)^*$ is a non-deterministic relation, and that not all sequences of refinement lead to a solved form. However, the set of possible solved forms reachable from σ completely characterizes the set of solutions of σ. Formally, for any symbolic trace σ, we let $\mathbf{SF}(\sigma) \stackrel{\text{def}}{=} \{\sigma' \mid \sigma\ (\succ)^*\ \sigma'$ and σ' is in sf$\}$. Then we have the following theorem

Theorem 13 (characterization of solutions). *Let σ be a symbolic trace and s a trace. Then s is a solution of σ if and only if s is a solution of some $\sigma' \in \mathbf{SF}(\sigma)$.*

By the above theorem and Lemma 9, we obtain:

Corollary 14. *A symbolic trace σ is consistent if and only if $\mathbf{SF}(\sigma) \neq \emptyset$.*

Note that $\mathbf{SF}(\sigma)$ can be effectively computed, and is always finite, as: (a) $\succ$ is finitely-branching relation, and (b) infinite sequences of refinement steps cannot arise. As to the latter point, note that, since each (REF_1)-step eliminates at least one variable, any sequence of refinement steps can contain only finitely many (REF_1)-steps, after the last of which rule (REF_2) can only be applied a finite number of times. Thus, computing $\mathbf{SF}(\sigma)$ gives a method to decide consistency of σ. This also suggests a method for solving STAP. As an example, suppose that we want to check the property $\sigma \models \bot \leftarrow \alpha$, that is, no solution of σ contains an instance of α. Then we can proceed as follows: for each action γ in σ, we check whether there is a mgu θ that unifies γ and α; if such a θ does not exist, or if it exists but $\sigma\theta$ is not consistent (i.e. $\mathbf{SF}(\sigma\theta) = \emptyset$, Corollary 14), then the property is true, otherwise it is not. Considering the general case $\alpha \leftarrow \beta$ leads us to the next theorem, which gives us an effective method to check $\sigma \models \alpha \leftarrow \beta$. Its proof relies on Theorem 13 and on Lemma 9, plus routine calculations on mgu's.

Theorem 15 (a method for STAP). *Let σ be a symbolic trace and let $pr = \alpha \leftarrow \beta$, where $\mathrm{v}(\alpha) \subseteq \mathrm{v}(\beta)$ and $\mathrm{v}(\beta) \cap \mathrm{v}(\sigma) = \emptyset$. Then $\sigma \models pr$, if and only if the following is true: for each θ such that $\alpha \in_\theta \mathrm{act}(\sigma)$ and for each $\sigma' \in \mathbf{SF}(\sigma\theta)$, say $\sigma' = \sigma\theta\theta'$, it holds that $\alpha\theta\theta'$ occurs prior to $\beta\theta\theta'$ in σ'.*

[1] Unmarking of $\hat{x}$ occurs in a (REF_1)-step if the prefix $\sigma\backslash\hat{x}$ gets shorter, like in: $\overline{\mathsf{a}}\langle\{a\}_k\rangle \cdot \mathsf{a}\langle\hat{z}\rangle \cdot \overline{\mathsf{a}}\langle\{\hat{z}\}_h\rangle \cdot \overline{\mathsf{a}}\langle k\rangle \cdot \mathsf{a}\langle\{\hat{x}\}_{\hat{y}}\rangle \cdot \mathsf{a}\langle\{\{\hat{x}\}_{\hat{y}}\}_h\rangle \succ \overline{\mathsf{a}}\langle\{a\}_k\rangle \cdot \mathsf{a}\langle\{x\}_y\rangle \cdot \overline{\mathsf{a}}\langle\{\{x\}_y\}_h\rangle \cdot \overline{\mathsf{a}}\langle k\rangle \cdot \mathsf{a}\langle\{x\}_y\rangle \cdot \mathsf{a}\langle\{\{x\}_y\}_h\rangle$, where $\{\{\hat{x}\}_{\hat{y}}\}_h$ gets unified with $\{\hat{z}\}_h$.

The above theorem immediately yields decidability of STAP, because there are finitely many mgu's θ to consider (at most one for each action in σ), and $\mathbf{SF}(\sigma)$ can be effectively computed. This result lifts of course to configurations.

Corollary 16 (decidability). *Let $\mathcal{C}$ be an initial configuration and $\alpha \hookleftarrow \beta$ be a property. It is decidable whether $\mathcal{C} \models \alpha \hookleftarrow \beta$ or not.*

Proof: Compute $\{\sigma \mid \mathcal{C}$ symbolically generates $\sigma\}$, which is finite, and then check whether or not for each σ in this set it is the case that $\sigma \models \alpha \hookleftarrow \beta$, which can be effectively done. The thesis is a consequence of Theorem 6. □

In a practical implementation, rather than generating the whole set of symbolic traces of a given configuration and then check the property, it is more convenient to check the single symbolic traces as soon as they are generated in an 'on-the-fly' way.

6 Restriction

We consider extending the base language via the *restriction* operator $(\mathsf{new}\, a)A$, where $a \in \mathcal{LN}$ and A an agent; $(\mathsf{new}\, a)$ is binder for name a. The intended meaning of $(\mathsf{new}\, a)A$ is that a new name a is created, which is private to A. The concrete and symbolic rules for restriction are given below. A function $\mathrm{new}_{\mathcal{LN}}(\cdot)$ is assumed that, for any set of names $V \subseteq_{\mathrm{fin}} \mathcal{LN}$, yields a local name $a \notin V$.

$$(\textsc{New})\ \langle s,\ (\mathsf{new}\, a)P\rangle \longrightarrow \langle s,\ P\rangle \qquad a = \mathrm{new}_{\mathcal{LN}}(V)$$
$$(\textsc{New}_{\mathrm{s}})\ \langle \sigma,\ (\mathsf{new}\, a)A\rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}} \langle \sigma,\ A\rangle_{\mathrm{s}} \quad a = \mathrm{new}_{\mathcal{LN}}(V)$$

In both rules, V is the set of local names occurring free in the source configuration. Note that the side-condition on name a is always met modulo alpha-renaming. A change is required in the rules for parallel composition $A \parallel B$, both in the concrete and in the symbolic case: in the conclusion, an additional renaming $[b/a]$ (where $a = \mathrm{new}_{\mathcal{LN}}(\ln(\sigma, A))$ and $b = \mathrm{new}_{\mathcal{LN}}(\ln(\sigma, A \parallel B))$) is applied onto the target configuration: this prevents a new name a possibly created by a (New)-transition of A from clashing with free occurences of a in B (this is just the side-condition of the rule (Par) of the π-calculus [17] rephrased in our language).

7 Conclusions

We have presented a symbolic method for analysing cryptographic protocols. The method is well suited for an efficient mechanization. The word 'efficient' should be taken with a grain of salt here. The trace analysis problem is obviouvsly NP-hard [4,10]. Pathological examples are therefore unavoidable, hence formal statements on 'efficiency' are difficult to formulate. However, we expect the method to perform well in practical cases (this is further discussed in the comparison with related work below). Experiments conducted with a preliminary, non optimized implementation have given encouraging results [6]. Developments

in the near future include an optimized implementation of the method and extension of the present results to other cryptographic primitives, like public key and hashing: this should not present conceptual difficulties, though we have not checked the details yet.

Approaches based on symbolic analysis have also been explored by Huima in [12] and Amadio and Lugiez [4]. In [12], Huima presents a symbolic semantics by which the execution of a protocol generates a set of equational constraints; only an informal description is given of the kind of equational rewriting needed to solve these constraints. Amadio and Lugiez in [4] consider a variant of the spi-calculus equipped with a symbolic semantics. Similarly to Huima's, their symbolic semantics generates equational constraints of a special form, rather than unifiers. The (rather complex) constraint-solving procedure is embedded into symbolic execution, and uses a brute-force method to resolve variables in key position (all possible instantiations of variables to names that are around are tried). These factors have a relevant impact on the size of the symbolic model. On the contrary, in our case symbolic execution and consistency check are kept separate, and this permits to keep the size of the model to a minimum. The consistency check procedure (refinement) is invoked only when necessary, and, most important, does not rely on brute-force instantiation. As a minor point, Amadio and Lugiez encode authentication via reachability: this may add to the complexity of their method.

Model checking [9,13,19,21] and theorem proving [18] seem to be among the most successful approaches to the formal analysis of security protocols. As Paulson has pointed out [18], theorem proving is intuitive, but, within it, verification is not fully automated and general completeness results are difficult to establish. On the contrary, model checking is automatic, but suffers from the state explosion problem, which requires the model to be cut down to a convenient finite size. Our paper might be regarded as an attempt at bridging the two approaches. We extract the unification mechanism underlying theorem proving and bring it on the ground of a process language (a variant of the spi-calculus) that naturally supports a notion of variable binding. This allows us to obtain precise completeness results for trace analysis.

Acknowledgements. I have benefitted from stimulating discussions with Martin Abadi, Roberto Amadio, Rocco De Nicola, Marcelo Fiore and Rosario Pugliese.

References

1. M. Abadi, A.D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1-70, 1999.
2. M. Abadi, A.D. Gordon. A Bisimulation Method for Cryptographic Protocols. *Nordic Journal of Computing*, 5(4):267-303, 1998.
3. R. Amadio, S. Prasad. The game of the name in cryptographic tables. RR 3733 INRIA Sophia Antipolis. In *Proc. of Asian'00*, LNCS, 2000.
4. R.M. Amadio, S. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. of Concur'00*, LNCS, 2000. Full version: RR 3915 Inria Sophia Antipolis.

5. M. Boreale. Symbolic analysis of cryptographic protocols in the spi-calculus. Manuscript, 2000. Available at `http://www.dsi.unifi.it/∼boreale/papers.html`.
6. M. Boreale. STA: a tool for trace analysis of cryptographic protocols. ML object code and examples, 2001. Available at `http://www.dsi.unifi.it/∼boreale/tool.html`.
7. M. Boreale, R. De Nicola, R. Pugliese. Proof Techniques for Cryptographic Processes. In *Proc. of LICS'99*, IEEE Computer Society Press, 1999. Full version to appear in *SIAM Journal on Computing.*
8. M. Burrows, M. Abadi, R.M. Needham. A logic of authentication. *Proc. of the Royal Society of London*, 426:233–271, 1989.
9. E.M. Clarke, S. Jha, W. Marrero. Using state exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. of the IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.
10. N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov. Undecidability of bounded security protocols. In *Proc. of Workshop on Formal Methods and Security Protocols*, Trento, 1999.
11. D. Dolev, A.C. Yao. On the security of public key protocols. In *IEEE Transactions on Information Theory* 29(2):198–208, 1983.
12. A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. of Workshop on Formal Methods and Security Protocols*, Trento, 1999.
13. G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Proceedings of TACAS'96*, (T. Margaria, B. Steffen, Eds.), *LNCS* 1055, pp. 147-166, Springer-Verlag, 1996.
14. G. Lowe. A Hierarchy of Authentication Specifications. In *10th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 1997.
15. G. Lowe. Towards a completeness result for model checking of security protocols. In *11th Computer Security Foundations Workshop*, IEEE Computer Society Press, 1998.
16. D. Marchignoli, F. Martinelli. Automatic verification of cryptographic protocols through compositional analysis techniques. In *Proc. of TACAS99*, LNCS 1579:148–163, 1999.
17. R. Milner, J. Parrow, D. Walker. A calculus of mobile processes, (Part I and II). *Information and Computation*, 100:1-77, 1992.
18. L.C. Paulson. Proving Security Protocols Correct. In *Proc. of LICS'99*, IEEE Computer Society Press, 1999.
19. A.W. Roscoe. Modelling and verifying key-exchange using CSP and FDR. In *8th Computer Security Foundations Workshop*, IEEE Computer Society Press, 1995.
20. A.W. Roscoe. Proving security protocols with model checkers by data independent techniques. In *11th Computer Security Foundations Workshop*, IEEE Computer Society Press, 1998.
21. S. Schneider. Verifying Authentication Protocols in CSP. *IEEE Transactions on Software Engineering*, 24(8):743-758, 1998.
22. S. Stoller. A reduction for automated verification of security protocols. In *Proc. of Workshop on Formal Methods and Security Protocols*, Trento, 1999.

Tree Automata with One Memory, Set Constraints, and Ping-Pong Protocols*

Hubert Comon[1,2], Véronique Cortier[2], and John Mitchell[1]

[1] Department of Computer Science, Gates 4B, Stanford University, CA 94305-9045
{comon,jcm}@theory.stanford.edu, Fax: (650) 725 4671
[2] Laboratoire Spécification et Vérification, CNRS and Ecole Normale Supérieure de Cachan,{comon,cortier}@lsv.ens-cachan.fr

Abstract. We introduce a class of tree automata that perform tests on a memory that is updated using function symbol application and projection. The language emptiness problem for this class of tree automata is shown to be in DEXPTIME. We also introduce a class of set constraints with equality tests and prove its decidability by completion techniques and a reduction to tree automata with one memory. Set constraints with equality tests may be used to decide secrecy for a class of cryptographic protocols that properly contains a class of memoryless "ping-pong protocols" introduced by Dolev and Yao.

1 Introduction

Set constraints were introduced in the eighties and have been studied thoroughly since, with applications to the analysis of programs of various styles (see [2] for a survey). Typically, the problem of interest is to decide the satisfiability of a conjunction of *set expression inclusions* $e \subseteq e'$ in which the set expressions are built from variables and various constructions, including, e.g., *projection*. Athough some set variable may occur several time in an expression, most classes of set constraints do not make it possible to write a set expression for a set of terms of the form $f(t,t)$, in which one subterm occurs more than once. One exception is the class of constraints studied in [6].

Our motivating interest is to develop classes of cryptographic protocols for which some form of secrecy is decidable. A historical class of decidable protocols are the so-called *ping-pong protocols* [10]. Although none of the protocols of [8] belongs to this class, ping-pong protocols remain a decidable class, while most larger classes of security protocols are undecidable [5]. One of the main restrictions in [11,10] is that messages are built using unary symbols only. In contrast, many protocols of interest are written using a binary encryption symbol and a pairing function. Another restriction in [11,10] is that each protocol participant is stateless: after a message is sent, the participant does not retain any memory of the contents of the message. This is a significant limitation since many protocols

* Partially supported by DoD MURI "Semantic Consistency in Information Exchange," ONR Grant N00014-97-1-0505, and NSF CCR-9629754.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 682–693, 2001.

rely on challenge-response steps, that require memory. A previous investigation of ping-pong protocols with added state led to undecidability [13].

It is insightful to observe that Dolev and Yao's result [11] can be proved using set constraints. This suggests a generalization of their approach to trees. A technical complication, though, is that the generalization to trees is less expressive than one might expect: in the case of unary functions only, a function and its inverse are set inverses of each other, in the sense that $f(f^{-1}(X))$ is precisely X. However, this is no longer true with trees: if f_1^{-1} and f_2^{-1} are the two projections corresponding to a binary function symbol f, the set $f(f_1^{-1}(X), f_2^{-1}(X))$ contains pairs $f(t_1, t_2)$ which are not necessarily in X. In order to increase the expressiveness of set constraints with binary functions, we need a "diagonal construction", enabling us to test for equalities the members of sets.

In this paper, we introduce a new class of set constraints, allowing limited diagonal constructions. This class is incomparable with the class sketched in [6]. We show that satisfiability is decidable for this class, allowing us to generalize Dolev and Yao's result to trees. Our class of set constraints does not capture all protocol concepts of interest. In particular, as can be seen from the survey [8], many authentication protocols make use of *nonces* or *time stamps*, which we cannot express. On the other hand, properties of protocols that are modeled using set constraints are decidable, while nonces and timestamps typically lead to undecidability [5]. Moreover, we can express conservative approximations of general protocols, and it is possible in principle that set constraints with equality tests provide algorithms for determining the security of some such protocols.

We prove the decidability of set constraints with equality tests by a reduction to an emptiness problem for a class of *tree automata with constraints*. Tree automata with various forms of constraints have been studied by several authors (see [9] for a survey). However, the class we consider in this paper is incomparable with known decidable classes. Roughly, we allow each state to hold one arbitrarily large memory register and restrict the use of this memory to equality tests. Since memory registers are updated using projections and function application, this class is a generalization of pushdown word (alternating) automata. Despite the generality of the class, there is a simple proof that emptiness decision is in DEXPTIME.

After discussing the correspondence between protocols and set constraints in the next section, we recall known results on definite set constraints which will be used later (section 3). Then we introduce tree automata with one memory in section 4; this section can be seen as a stand-alone decidability result and relies on definite set constraints. Next, we introduce in section 5 our class of set constraints with one equality, showing how to reduce the satisfiability of these constraints to the non-emptiness decision for tree automata with one memory. The reduction is similar to the saturation process described in [7] for set constraints with intersection, but it is slightly more complicated due to equality tests. (In fact, we obtain a doubly exponential algorithm if the maximum arity of all function symbols is not constant.) Finally, we discuss the application to security protocols in section 6.

2 Protocol Motivation

Dolev and Yao [11] consider protocols in which each principal holds a single public key (which is known to everybody) and a corresponding private key that is known to them only. The principals are able to build messages using plain text, encryption e_X with the public key of X and signatures d_X appending the name of principal X. Here is a simple example from [11]:

$A \to B : e_B(d_A(e_B(s)))$	Alice sends to Bob a message encrypted using Bob's public key consisting of a signed encrypted text s
$B \to A : e_A(s)$	Bob acknowledges the reception by sending back to Alice the text s, encrypted using the public key of Alice

In this model, communication channels are insecure. This allows an intruder to intercept messages, remember them, and replace them with alternate (possibly forged) messages. The intruder may decrypt a message if the corresponding key has become known to him, may append or remove signatures, and may encrypt using any public key. The secrecy question asks whether there is a way for an intruder to get the plain text message s that is supposed to be kept secret between Alice and Bob. In the above example, the answer is yes (the protocol is insecure).

The possible use of set constraints in cryptographic protocols analysis has been suggested in several papers, e.g. [14]. It is however interesting to see that the Dolev-Yao decidability proof can be summarized using set constraints by letting I be the set of messages that can be built by the intruder (after any number of sessions). Since I can intercept any message of any run of the protocol, we write set constraints putting every protocol message in I. For the example protocol above, we have

$$e_Y(d_X(e_Y(s))) \subseteq I \qquad\qquad e_X(e_Y^{-1}(d_X^{-1}(e_Y^{-1}(I)))) \subseteq I$$

for every pair of principals X, Y, since Bob acknowledges a message m from Alice by sending $e_A(e_B^{-1}(d_A^{-1}(e_B^{-1}(m))))$. Finally, for every principal X, we express the ability of the intruder to perform operations using public information about X:

$$d_X(I) \subseteq I, \;\; e_X(I) \subseteq I, \;\; d_X^{-1}(I) \subseteq I$$

This process translates a protocol into a collection of set constraints about the set I of messages available to the intruder. Secrecy now becomes the question whether the set constraints, together with $s \notin I$, is satisfiable ? Assuming a fixed number of principals, this is decidable in polynomial time for set constraints arising from Dolev-Yao's ping-pong protocols: we can compute the minimal solution of the definite set constraint and check the membership of s.

There are several restrictions in the Dolev-Yao approach. In particular, only a fixed number of principals and, as mentioned above, only unary symbols may be used. A pairing function or a binary encryption symbol, allowing to write e.g. $e(k, m)$ instead of $e_k(m)$, i.e. allowing to consider keys as first-class objects, would considerably increase the expressive power.

3 Definite Set Constraints

This class of set constraints has been introduced in [15] and studied by various authors (e.g. [7]). Each constraint is a conjunction of inclusions $e_1 \subseteq e_2$ where e_1 is a *set expression* and e_2 is a *term set expression*. Term set expressions are built out of a fixed ranked alphabet of function symbols $\mathcal{F}$, the symbol $\top$ and set variables. A set expression is either a term set expression or a union of two set expressions $e_1 \cup e_2$, or an intersection of two set expressions $e_1 \cap e_2$ or the image of set expressions by some function symbol $f(e_1, \ldots, e_n)$ or a projection $f_i^{-1}(e_1)$ where f is a function symbol and $i \in [1..n]$ if n is the rank of f. Note that negation is not allowed. Here is a definite set constraint:

$$f_2^{-1}(X) \subseteq g(Y) \qquad f(f(X,Y) \cap X, X) \subseteq X \qquad g(Y) \cap Y \subseteq X \qquad a \subseteq Y$$

Set expressions denote sets of subsets of the Herbrand universe $T(\mathcal{F})$; if σ assigns each variable to some subset of $T(\mathcal{F})$, then $[\![\ ]\!]_\sigma$ is defined by:

$$\begin{array}{ll} [\![X]\!]_\sigma \stackrel{\text{def}}{=} X\sigma & [\![f(e_1,\ldots,e_n)]\!]_\sigma \stackrel{\text{def}}{=} \{f(t_1,\ldots,t_n) \mid \forall i \in [1..n], t_i \in [\![e_i]\!]_\sigma\} \\ [\![e_1 \cap e_2]\!]_\sigma \stackrel{\text{def}}{=} [\![e_1]\!]_\sigma \cap [\![e_2]\!]_\sigma & [\![f_i^{-1}(e)]\!]_\sigma \stackrel{\text{def}}{=} \{t_i \mid \exists t_1, ..., t_n . f(t_1, ..., t_n) \in [\![e]\!]_\sigma\} \\ [\![\top]\!]_\sigma \stackrel{\text{def}}{=} T(\mathcal{F}) & [\![e_1 \cup e_2]\!]_\sigma \stackrel{\text{def}}{=} [\![e_1]\!]_\sigma \cup [\![e_2]\!]_\sigma \end{array}$$

σ satisfies $e_1 \subseteq e_2$ iff, $[\![e_1]\!]_\sigma \subseteq [\![e_2]\!]_\sigma$. This extends to conjunctions of inclusions.

Theorem 1 ([7]). *The satisfiability of definite set constraints is DEXPTIME-complete and each satisfiable constraint has a least solution.*

4 Tree Automata with One Memory

The idea is to enrich the expressiveness of tree automata by allowing them to carry and test some information. For instance, a pushdown automaton will keep a stack in its memory and check the symbols at the top of the stack. What we do here is something similar. Our automata work on trees instead of words and may perform more general constructions and more general tests.

Informally, a tree automaton with one memory computes bottom-up on a tree t by synthesizing both a state (in a finite set of states Q) and a memory, which is a tree over some alphabet Γ. Each transition uses some particular function which computes the new memory from the memories at each direct son. Each transition may also check for equalities the contents of the memories at each son.

Given an alphabet of function symbols Γ, the set of functions Φ which we consider here (and which may be used to compute on memories) is the least set of functions over $T(\Gamma)$ which is closed by composition and containing:

- for every $f \in \Gamma$ of arity n, the function $\lambda x_1, ...x_n.f(x_1, \ldots, x_n)$
- for every n and every $1 \leq i \leq n$, the function $\lambda x_1, ..., x_n.x_i$
- for every $f \in \Gamma$ of arity n and for every $1 \leq i \leq n$, the (partial) function which associates each term $f(t_1, \ldots, t_n)$ with t_i, which we write $\lambda f(\boldsymbol{x}).x_i$.

For instance, if Γ contains a constant (empty stack) and unary function symbols, Φ is the set of functions which push or pop after checking the top of the stack.

Definition 1. *A tree automaton with one memory is a tuple* $(\mathcal{F}, \Gamma, Q, Q_f, \Delta)$ *where* $\mathcal{F}$ *is an alphabet of input function symbols,* Γ *is an alphabet of memory function symbols,* Q *is a finite set of states,* Q_f *is a subset of final states,* Δ *is a finite set of transition relations of the form* $f(q_1, ..., q_n) \xrightarrow[F]{c} q$ *where*

- $f \in \mathcal{F}$ *is called the* head symbol of the rule,
- c *is a subset of* $\{1, ..., n\}^2$, *defining an equivalence relation on* $\{1, \ldots, n\}$.
- $\lambda x_1 ..., x_k F(x_1, \ldots, x_k) \in \Phi$, *where* k *is the number of classes modulo* c
- $q_1, \ldots, q_n, q \in Q$, *(*$q$ *is the* target *of the rule).*

A *configuration* of the automaton consists of a state and a term in $T(\Gamma)$ (the memory). Then computations work as follows: if $t = f(t_1, \ldots, t_n)$ and the computation on $t_1, \ldots, t_n$ respectively yields the configurations $q_1, \tau_1, \ldots, q_n, \tau_n$, then the automaton, reading t, may move to q, τ when there is a transition rule $f(q_1, \ldots, q_n) \xrightarrow[F]{c} q$ and for every $i = j \in c$, $\tau_i = \tau_j$ and $\tau = F(\tau_{i_1}, \ldots, \tau_{i_k})$ where $i_1, \ldots, i_k$ are representatives of the equivalence classes for c. A tree t is accepted by the automaton whenever there is a computation of the automaton on t yielding a configuration q, γ with $q \in Q_f$.

Example 1. Assume that the transitions of the automaton A are (other components of the automaton are obvious from the context, $\top$ is the identity relation):

$$g(q) \xrightarrow[\lambda x_1.x_1]{\top} q \qquad f(q_a, q_a) \xrightarrow[\lambda x_1.h(x_1)]{1=2} q \qquad a \xrightarrow[b]{\top} q_a$$
$$g(q_a) \xrightarrow[\lambda x_1.h(x_1)]{\top} q \qquad f(q, q) \xrightarrow[\lambda h(x_1).x_1]{1=2} q$$

A computation of the automaton on $f(g(f(a, a)), g(a))$ is displayed on figure 1, in which the configurations reached at each node are displayed in a frame.

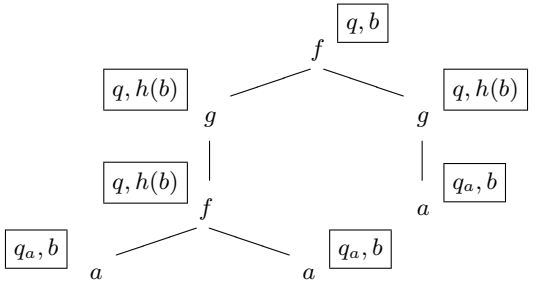


Fig. 1. A tree t and a computation of A on t

Pushdown automata (on words) perform transitions $a, q, \alpha \cdot \gamma \to q', \beta \cdot \gamma$ where a is an input symbol, q, q' are states and α, β, γ are words over the stack alphabet (the rule pops α and pushes β). Such a rule can be translated in the above formalism, viewing letters as unary symbols: $a(q) \xrightarrow[\lambda x.\beta\alpha_1^{-1}x]{} q'$. This

translation does not make use of equality tests. Orthogonally, if we use the tests, but assume that $F = \lambda \boldsymbol{x}.f(\boldsymbol{x})$ for each rule headed with f, then we get tree automata with equality tests between brothers (see [4]).

Theorem 2. *The emptiness of the language recognized by a tree automaton with one memory is decidable in DEXPTIME. More generally, the reachability of a given configuration is decidable in DEXPTIME.*

Proof. (sketch) For every $q \in Q$, let M_q be the subset of $T(\Gamma)$ of memory contents m such that there is a tree t and a computation of the automaton on t yielding the configuration $< q, m >$. Then the sets M_q are the least solutions of the definite set constraint, consisting, for each transition rule $f(q_1, \ldots, q_n) \xrightarrow[F]{c} q$ of the inclusion $F(L_{q_{i_1}}, ...L_{q_{i_k}}) \subseteq M_q$ and $L_{q_{i_j}}$ is the intersection for all indices l equivalent (w.r.t. c) to i_j of M_l. Then the non-emptiness of the language (resp. reachability of a configuration) reduces to similar questions on definite set constraints, which are solvable in DEXPTIME.

The result can be generalized to alternating tree automata with one memory keeping the same complexity. Alternation here has to be understood as follows: we may replace the states occurring in the left hand sides of the rules with arbitrary positive Boolean combinations of states. The above proof simply works, using additional intersections and unions.

Corollary 1. *The emptiness problem of alternating tree automata with one memory is DEXPTIME-complete.*

Note however that the class of automata with one memory is neither closed under intersection nor complement (both yield undecidable models).

5 Set Constraints with Equality Tests

We consider now definite set constraints as in section 3 with an additional construction: function symbols can be labeled with equality tests, which are conjunctions of equalities $p_1 = p_2$ between paths. The intention is to represent sets of terms t such that the subterms at positions p_1 and p_2 are identical.

More precisely, if c is a conjunction of equalities between paths (which we assume w.l.o.g. closed under transitivity and such that no strict prefix of a path in c is in c), we define

$$[\![f^c(e_1, \ldots, e_n)]\!]_\sigma \stackrel{\text{def}}{=} \{t \in [\![f(e_1, \ldots, e_n)]\!]_\sigma \mid t \models c\}$$

and $t \models c$ if, for every equality $p_1 = p_2$ in c, p_1, p_2 are positions in t and $t|_{p_1} = t|_{p_2}$. If $p_1 = p_2 \in c$, we say thatp_1, p_2 *are checked by* c. All other constructions are the same as in section 3. In particular, right hand sides of inclusions should not contain constructions f^c. When c is empty, we may omit it or write $\top$.

Example 2. $f^{21=12}(f(Z,Y)\cap X, g(X)\cap Y) \subseteq f(Y,X)$ is an inclusion constraint. $\sigma = \{X \mapsto \{a,b,f(a,b)\}; Y \mapsto \{b,g(a),g(b),f(a,b)\}; Z \mapsto \{a,b\}\}$ is a solution of the constraint since $[\![f^{12=21}(f(Z,Y)\cap X, g(X)\cap Y)]\!]_\sigma = \{f(f(a,b),g(b))\}$

As a consequence of undecidability results on tree automata with equality tests (see e.g. [9]), the satisfiability of such general constraints is undecidable. That is why we are going to put more restrictions on the constraints.

If X is a variable of a constraint S, then let $\mathcal{R}(X)$ be the set of atomic constraints whose right hand side contains X. The set of variables having a *basic type* is the least set of variables X such that $\mathcal{R}(X)$ consists of inclusions $g_i(X_1^i,\ldots,X_{n_i}^i) \subseteq X$ and such that

- if the symbols g_i do not occur anywhere else in S and every X_j^i is either X itself or has a basic type, then X has a basic type
- if every X_j^i has a basic type, then X has a basic type.

Intuitively, the basic types correspond to data whose format is irrelevant (first case in the definition) or which can be built using a bounded number of symbols on such data (second case). For example, `Nat` and `U` have basic types in

$$\mathcal{R}(\texttt{Nat}) \cup \mathcal{R}(\texttt{U}) \stackrel{\text{def}}{=} \quad \texttt{zero} \subseteq \texttt{Nat}, \qquad s(\texttt{Nat}) \subseteq \texttt{Nat}, \qquad pn(\texttt{Nat}) \subseteq \texttt{U}$$

if `zero` and s are not used elsewhere in S.

This notion is extended to expressions: an expression e is *basic* if

- e is a basic variable or
- e is an intersection $e_1 \cap e_2$ and either e_1 or e_2 is basic
- e is an expression $e_1 \cup e_2$ and both e_1 and e_2 are basic
- e is an expression $f(e_1,\ldots,e_n)$ (or $f^c(e_1,\ldots,e_n)$) and $e_1,\ldots,e_n$ are basic

The set of *paths* in an expression e is defined as follows: $\Pi(f^c(e_1,\ldots,e_n)) \stackrel{\text{def}}{=} 1\cdot\Pi(e_1)\cup\ldots\cup n\cdot\Pi(e_n)$, $\Pi(e_1\cap e_2) \stackrel{\text{def}}{=} \Pi(e_1\cup e_2) \stackrel{\text{def}}{=} \Pi(e_1)\cup\Pi(e_2)$, $\Pi(f_i^{-1}(e)) \stackrel{\text{def}}{=} \emptyset$. $e|_p$ is any of the subexpressions at position p. When there is no $\cup$ or $\cap$ symbol along a path p then $e|_p$ denotes a single expression.

The assumption: We assume that, in each subexpression $f^c(e_1,\ldots,e_n)$, for every $p_1 = p_2 \in c$, p_1 and p_2 are paths of $f(e_1,\ldots,e_n)$. (This is actually equivalent to restricting the use of projections below an f^c construction). Then, for each expression $f^c(e_1,...,e_n)$ we require that, if $p\cdot i\cdot q$ is checked by c and p is not empty then, for $i \neq j$, either $p\cdot j = p\cdot i\cdot q \in c$ or any subexpression at position $p\cdot j$ has a basic type or any subexpression at $p\cdot i\cdot q$ has a basic type. This will be referred to as the *basicness condition.*

Example 3. If c is $12 = 21 = 11 \wedge 22 = 3$. the basicness condition imposes that either $e|_{21}$ or $e|_{22}$ has a basic type (and hence the other expressions at equivalent positions).

The resulting constraints are called *set constraints with equality tests* (*ET-constraints* for short). We can construct an ET-constraint whose least solution is the set of trees $\Delta = \{f(t,t) \mid t \in T(\mathcal{F})\}$. The only other decidable set constraint formalism which allows to express Δ is the class defined in [6], in which, however, equality tests are restricted to brother positions (which is not the case here). On the other hand, we have restrictions which are not present in [6].

5.1 Saturation

We use here a fixed point computation method which is similar to the one in [7]: the goal is to deduce enough consequences so that the inclusions whose right hand side is not a variable become redundant, hence can be discarded. Unfortunately, the first step (representation) in [7] cannot be used in the same way here, since it does not preserve the class of constraints we consider. Still, as a first step, we can get rid of projections and unions: we can compute in plynomial time an equivalent ET-constraints containing no union or projection.

Next, we normalize the expressions according to the following rule **Norm**:

$$f^c(e_1, \ldots, e_n) \cap f^{c'}(e'_1, \ldots, e'_n) \to f^{c \wedge c'}(e_1 \cap e'_1, \ldots, e_n \cap e'_n)$$

Lemma 1. *(***Norm***) transforms an expression in an equivalent one. Moreover, if the basicness condition is satisfied by the premises, then there is a (effectively computable) constraint c'' which is logically equivalent to $c \wedge c'$ and such that $f^{c''}(e_1 \cap e'_1, \ldots, e_n \cap e'_n)$ satisfies the basicness condition.*

We can get rid of basic type variables:

Lemma 2. *For each basic variable X, there is a ground expression t_X such that, if S' is the ET-constraint obtained by replacing in S each basic variable X with t_X, then σ is a solution of S' iff $\sigma \circ \sigma_X$ is a solution of S.*

We may assume now that there is no basic variable in the constraint. Then the tests can be simplified, removing positions of basic expressions.

Next, as in [7], we extend the language allowing non-emptiness preconditions in the rules, which allow to simplify (some but not all) inclusion constraints $f^c(\boldsymbol{e}) \subseteq f(\boldsymbol{e'})$. Formally, the set constraints are now clauses of the form

$$\mathsf{nonempty}(e_1), \ldots, \mathsf{nonempty}(e_n) \Rightarrow e \subseteq e'$$

where $e, e_1, \ldots, e_n$ are set expressions,e' is a set expression using only intersections, variables and function symbols (without constraints). $\mathsf{nonempty}(e_i)$ is satisfied by an assignment σ iff $[\![e_i]\!]_\sigma$ is not empty.

Then we remove constraints of the form $\phi \Rightarrow C[f^c(\boldsymbol{e}) \cap g^{c'}(\boldsymbol{e'})] \subseteq e''$ and replace constraints $\phi \Rightarrow e \subseteq C[f(\boldsymbol{e}) \cap g(\boldsymbol{e'})]$ with $\phi, \mathsf{nonempty}(e) \Rightarrow \mathbf{false}$ for every context C and every $f \neq g$. These rules are correct, by definition of the interpretation.

We also abstract out subexpressions introducing new variables, as long as this preserves the form of the constraints. For instance, for contexts $C[\ ]_p$, an inclusion $C[f^c(\boldsymbol{e})]_p \subseteq e'$ becomes $C[X]_p \subseteq e', f^c(\boldsymbol{e}) \subseteq X$ where X is a new variable. This results in an equivalent constraint (on the original variables) in which the inclusions are $e \subseteq e'$ where e' is either an intersection of variables $X_1 \cap \ldots \cap X_n$ or an expression $f(X_1, \ldots, X_n)$ and e is either an intersection of variables or an expression $f^c(\boldsymbol{e})$ in which, at any position which is not a strict prefix of a position checked by c, there is a (non-basic) variable or a term t_X.

In addition, each time $p_1 = p_2$ appears in c in an expression $f^c(\boldsymbol{e})$, we assume that all subexpressions at positions p_1, p_2 are identical, which can be easily ensured replacing both subexpressions with their intersection.

Transitivity 1 $\dfrac{\phi_1 \Rightarrow e_1 \subseteq e_2 \quad \phi_2 \Rightarrow e_2 \subseteq e_3}{\phi_1, \phi_2 \Rightarrow e_1 \subseteq e_3}$

Transitivity 2 $\dfrac{\phi \Rightarrow f^c(\boldsymbol{e})[X \cap g(\boldsymbol{e_1}) \cap e_2]_p \subseteq e_3 \quad \phi' \Rightarrow g^{c'}(\boldsymbol{e_4}) \subseteq X}{\phi, \phi' \Rightarrow f^c(\boldsymbol{e})[g^{c'}(\boldsymbol{e_4}) \cap g(\boldsymbol{e_1}) \cap e_2]_p \subseteq e_3}$ provided p is a strict prefix of a path checked in c

Compatibility $\dfrac{\phi \Rightarrow e_1 \subseteq e_2 \quad \phi' \Rightarrow e_1' \subseteq e_2'}{\phi, \phi' \Rightarrow e_1 \cap e_1' \subseteq e_1' \cap e_2'}$

Clash $\dfrac{\phi \Rightarrow f(\boldsymbol{e}) \subseteq g(\boldsymbol{e'})}{\phi \Rightarrow \mathbf{false}}$ if $f \neq g$

Weakenings $\dfrac{\phi \Rightarrow e_1 \subseteq e_2 \cap e_3}{\phi \Rightarrow e_1 \subseteq e_2} \qquad \dfrac{\phi \Rightarrow e_1 \subseteq e_2}{\phi \Rightarrow e_1 \cap e_3 \subseteq e_2}$ If e_3 is an expression occurring somewhere in the set of constraints

Projection $\dfrac{\phi \Rightarrow f^c(e_1, \ldots, e_n) \subseteq f(e_1', \ldots, e_n')}{\phi, \mathsf{nonempty}(f^c(e_1, \ldots, e_n))) \Rightarrow e_i^{c\downarrow i} \subseteq e_i'}$ If the subexpression at every strict prefix of a position checked in c is of the form $g(\boldsymbol{e''})$ for some g.

Fig. 2. The saturation rules

Now, we are ready to apply the deduction rules given in figure 2, applying again abstractions and normalisation (eagerly) if necessary to keep the special form of the constraint. We use $e[e']_p$ to express either that e' is replaced by e at position p or that e' has to occur at position p. This means in particular that the subexpression at position p in e has to be defined in a unique way. $c \downarrow_i$ is defined by $(c \wedge c') \downarrow_i \stackrel{\text{def}}{=} c \downarrow_i \wedge c' \downarrow_i$, $(i \cdot p = i \cdot q) \downarrow_i \stackrel{\text{def}}{=} p = q$ and $(j \cdot p = q) \downarrow_i \stackrel{\text{def}}{=} \top$ when $i \neq j$. e^c is the expression in which the top symbol of e is constrained by c. (It is used only in a context where e must be headed with a function symbol or $c = \top$).

Lemma 3. *The inference rules in figure 2 are correct: the new constraint is a consequence of the previous ones.*

Lemma 4. *The rules of figure 2 are terminating: a fixed point is reached after finitely many steps (at most $O(2^{|S|\times|c|\times b\times 2^a})$ where a is the maximal arity of a function symbol, b is the number of basic types and $|c|$ is the maximal depth of an equality test).*

If S is an ET-constraint, let solved(S) be the clauses $\phi \to a$ in S such that either a is **false** or else a is an inclusion $f^c(\boldsymbol{e}) \subseteq e'$ where e' is an intersection of variables and $f^c(\boldsymbol{e})$ does not contain any subexpression of the form $X \cap g^c(\boldsymbol{e'})$ where X is a variable. Using a classical construction, we can show that:

Lemma 5. *solved(S) is either unsatisfiable or has a least solution.*

As in [7], the following completeness result is obtained by inspecting each clause $C \in S$ which is not in solved(S), showing that, thanks to saturatedness, the least solution of solved(S) is a solution of C. There are only some additional cases for non-flat constraints e.g. $f^c(X \cap g(\boldsymbol{e}), \boldsymbol{e'}) \subseteq f(\boldsymbol{e''})$.

Theorem 3. *If S is saturated, then either both S and solved(S) are unsatisfiable or else S has a least solution, which is the least solution of solved(S).*

5.2 The Main Result

We build, for each ET-constraint S, an automaton with one memory A_S such that if α is the least solution of S, for very variable X, $\alpha(X)$ is a simple homomorphic image of the set of terms accepted by A_S in state q_X. [1] The memory alphabet of the automaton is the set of function symbols used in the constraint and the alphabet $\mathcal{F}$ is the memory alphabet with some additional symbols allowing to check on auxilliary branches non emptiness conditions.

The set of states is the set of subexpressions occurring in the constraint, together with some local memory. We keep in the (unbounded) memory attached to each state the tree which will be checked later for equality. The idea is to accept in state e a term t iff there is at least one term in $[\![e]\!]_\sigma$ for every solution σ of the constraint. We have no room here to detail the construction.

As a consequence of lemma 3, lemma 4, theorem 2, theorem 3 and the above construction, we get:

Theorem 4. *The satisfiability of ET-constraints is decidable. Furthermore, given a set constraint with equality tests S, a term t and a free variable X of S, the consistency of $S \wedge t \notin X$ is equivalent to the non-reachability of the configuration q_X, t in A_S, which is decidable.*

[1] Note that we could prove the result directly, using the same trick as in the proof of theorem 2, without any reference to automata with one memory. However, we believe that theorem 2 is interesting in itself.

6 Analysis of Cryptographic Protocols

We sketch here a protocol example (inspired by Kerberos) that can be analyzed using ET-constraints, but is beyond the scope of [11]. We use a tupling function $< , >$, a binary encryption $\mathtt{enc}$ and several additional symbols.

1. $A \to S : A, B$
2. $S \to A : \mathtt{enc}(k(A), < B, K(A,B), \mathtt{enc}(k(B), < A, K(A,B) >) >)$
3. $A \to B : \mathtt{enc}(K(A,B), m(A,B)), \mathtt{enc}(k(B), < A, K(A,B) >)$
4. $B \to A : \mathtt{enc}(K(A,B), h(m(A,B)))$

In words, A tells the key server S that she wants to securely communicate with B. Then S sends back to A a message, encrypted using A's public key and containing a session key $K(A,B)$ together with a certificate which can be opened by B only. At the third step, A sends her message $m(A,B)$, encrypted using the key $K(A,B)$, together with the certificate, which is copied blindly from message 2. Finally, B acknowledges the reception, sending back a digest $h(m(A,B))$ of the previous message, encrypted using the shared key.

Because of lack of space, we only sketch here very briefly how to express the intruder capabilities on one hand and how the equality tests are used.

As in section 2, the intruder's knowledge is represented using a set variable I. We use a set variable S_i resp. R_i) for the set of messages sent (resp. received) at step i (see e.g. [12] for more details). Since the intruder I potentially intercepts all messages and may send any message he can forge, we have $S_i \subseteq I \subseteq R_i$. In addition, he has some capabilities to alter messages. For instance:

$$\mathtt{enc}(I, I) \subseteq I \qquad\qquad \mathtt{enc}(I, \top) \cap I \subseteq \mathtt{enc}(\top, I)$$

In words, I can encrypt a known message using a known key and I can decrypt a message whose encrypting key is known to him.

Now comes the protocol-dependent part of the constraints. The memory which is kept by the participants is modeled by sending at step i not only the ith message, but also relevant previous messages. This trick does not change anything to I's knowledge as previous messages were known to him. For instance, step 2 will consist of:

$$< R_1 \cap < \mathtt{U}, \mathtt{U} >, \mathtt{enc}(k(\mathtt{U}), < \mathtt{U}, K(\mathtt{U},\mathtt{U}), \mathtt{enc}(k(\mathtt{U}), < \mathtt{U}, K(\mathtt{U},\mathtt{U}) >) >) >^{c} \subseteq S_2$$

where $c \stackrel{\text{def}}{=} 11 = 211 = ... \wedge 12 = 221 = ...$ expresses that principal names do match. This is necessary since the intruder may be a participant in another session of the protocol: we would get a too rough overapproximation without these tests. At step 3, A includes blindly a piece of message 2, which can be expressed using an equality test on non-basic variables of the form: $< R_2 \cap \mathtt{enc}(..., ...X...), X >^{c} \subseteq S_3$ where c restricts the interpretations of the left hand side to sets of terms $< \mathtt{enc}(..., ...t...), t >$. The last message is successful only if B answers correctly the challenge, which can be expressed using an equality constraint representing A's memory: $< R_4 \cap \mathtt{enc}(X, h(Y)), S_3 \cap \mathtt{enc}(X, Y) >^{c} \subseteq S_5$ where c restricts the instances of the expression to sets of terms $< \mathtt{enc}(t_1, h(t_2)), \mathtt{enc}(t_3, t_2) >$.

We can handle an unbounded number of principals and messages may be built using any set of function symbols with any arity. However, we cannot handle nonces (randomly generated numbers) in general. In this respect, our decidability result is not more general than e.g. [3].

Considering nonces introduces several complications (which we can expect [12]): first we have to ensure that all nonces are distinct. This is possible at the price of introducing disequality tests on basic types in the set constraints, hence disequality tests in the automata. This may yield a still decidable model. A much harder issue is the freshness of the nonces. Indeed, each nonce has a lifetime, hence a scope (this becomes quite clear in spi-calculus formalizations [1]). Modeling the scope hardly fits into the (finite) set constraints formalism.

References

1. M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1), 1999.
2. A. Aiken. Introduction to set constraint-based program analysis. *Science of Computer Programming*, 35:79–111, 1999.
3. R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. CONCUR'00*, volume 1877 of *Lecture Notes in Computer Science*, 2000.
4. B. Bogaert and S. Tison. Equality and disequality constraints on brother terms in tree automata. In A. Finkel, editor, *Proc. 9th. Symposium on Theoretical Aspects of Comp. Science*, Cachan, France, 1992.
5. I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In P. Syverson, editor, *12-th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.
6. W. Charatonik and L. Pacholski. Negative set constraints with equality. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 128–136, Paris, 1994.
7. W. Charatonik and A. Podelski. Set constraints with intersection. In *Proc. IEEE Symposium on Logic in Computer Science*, Varsaw, 1997.
8. J. Clarke and J. Jacobs. A survey of authentication protocol. literature: Version 1.0. Draft paper, 1997.
9. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997.
10. D. Dolev, S. Even, and R. Karp. On the security of ping pong protocols. *Information and Control*, 55:57–68, 1982.
11. D. Dolev and A. Yao. On the security of public key protocols. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 350–357, 1981.
12. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Workshop on formal methods in security protocols*, Trento, Italy, 1999.
13. S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. Technical Report 285, Technion, Haifa, Israel, 1983. Extended abstract appeared in IEEE Symp. Foundations of Computer Science, 1983.
14. N. Heintze and J. Tygar. A model for secure protocols and their compositions. *IEEE transactions on software engineering*, 22(1), 1996.
15. N. Heinze and J. Jaffar. A decision procedure for a class of set constraints. In *Proc. IEEE Symp. on Logic in Computer Science*, Philadelphia, 1990.

Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata

Kousha Etessami[1], Thomas Wilke[2], and Rebecca A. Schuller[3]

[1] Bell Labs, Murray Hill, NJ
kousha@research.bell-labs.com
[2] Christian Albrecht University, 24098 Kiel, Germany
wilke@ti.informatik.uni-kiel.de
[3] Cornell University, Ithaca, NY
reba@math.cornell.edu

Abstract. We give efficient algorithms, beating or matching optimal known bounds, for computing a variety of simulation relations on the state space of a Büchi automaton. Our algorithms are derived via a unified and simple parity-game framework. This framework incorporates previously studied notions like *fair* and *direct* simulation, but our main motivation is state space reduction, and for this purpose we introduce a new natural notion of simulation, called *delayed* simulation. We show that, unlike fair simulation, delayed simulation preserves the automaton language upon quotienting, and that it allows substantially better state reduction than direct simulation. We use the parity-game approach, based on a recent algorithm by Jurdzinski, to efficiently compute all the above simulation relations. In particular, we obtain an $O(mn^3)$-time and $O(mn)$-space algorithm for computing both the delayed and fair simulation relations. The best prior algorithm for fair simulation requires time $O(n^6)$ ([HKR97]).
Our framework also allows one to compute bisimulations efficiently: we compute the fair bisimulation relation in $O(mn^3)$ time and $O(mn)$ space, whereas the best prior algorithm for fair bisimulation requires time $O(n^{10})$ ([HR00]).

1 Introduction

There are at least two distinct purposes for which it is useful to compute simulation relationships between the states of automata: (1) to efficiently establish language containment among nondeterministic automata; and (2) to reduce the state space of an automaton by obtaining its quotient with respect to the equivalence relation underlying the simulation preorder.

For state machines without acceptance conditions, there is a well-understood notion of simulation with a long history (see, e.g., [Mil89,HHK95]). For ω-automata, where acceptance (fairness) conditions are present, there are a variety of different simulation notions (see, e.g.,[HKR97],[GL94]). At a minimum, for such a simulation to be of use for purpose (1), it must have the following property:

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 694–707, 2001.

(*) whenever state q' "simulates" state q the language of the automaton with start state q' contains the language of the automaton with start state q.

This property alone however is not, as we will see in Sect. 5, sufficient to assure usefulness for purpose (2), which requires the following stronger property:

(**) the "simulation quotient" preserves the language of the automaton.

We will state precisely what is meant by a simulation quotient later.

In [HKR97] a number of the different simulation notions for ω-automata were studied using a game-theoretic framework. The authors also introduced a new natural notion of simulation, titled *fair* simulation. They showed how to compute fair simulations for both Büchi and, more generally, Streett automata. For Büchi automata, their algorithm requires $O(n^6)$ time to determine, for one pair of states (q, q'), whether q' fairly simulates q.[1] Their algorithm relies on an algorithm for tree automaton emptiness testing developed in [KV98]. In this paper, we present a new comparatively simple algorithm for Büchi automata. Our algorithm reduces the problem to a parity game computation, for which we use a recent elegant algorithm by Jurdzinski, [Jur00], along with some added enhancements to achieve our bounds. Our algorithm determines in time $O(mn^3)$ and space $O(mn)$ *all* such pairs (q, q') of states in an input automaton A where q' simulates q. Here m denotes the number of transitions and n the number of states of A. In other words, our algorithm computes the entire maximal fair simulation relation on the state space in the stated time and space bound.[2]

In [HKR97] the authors were interested in using fair simulation for purpose (1), and thus did not consider quotients with respect to fair simulation. The question arises whether fair simulation can be used for purpose (2), i. e., whether it satisfies property (**). We give a negative answer by showing that quotienting with respect to fair simulation fails badly to preserve the underlying language, under any reasonable definition of a quotient. On the other hand, there is an obvious and well known way to define simulation so that quotients do preserve the underlying language: *direct* simulation[3] ([Mil89,DHWT91]) simply accommodates acceptance into the standard definition of simulation by asserting that only an accept state can simulate another accept state. Direct simulation has already been used extensively (see, e.g., [EH00,SB00]) to reduce the state space of automata. Both [EH00] and [SB00] describe tools for optimized translations from linear temporal logic to automata, where one of the key optimizations is simulation reduction. However, as noted in [EH00], direct simulation alone is not able to reduce many obviously redundant state spaces. Recall that, in general, it is PSPACE-hard to find the minimum equivalent automaton for a given

[1] There is a typo in the original version of [HKR97] that indicates an $O(n^4)$ running time for their algorithm, but that typo has since been corrected.

[2] D. Bustan and O. Grumberg [BG00] have independently obtained an algorithm for computing fair simulation which, while it does not improve the $O(n^6)$ time complexity of [HKR97], improves the space complexity to $O(n^2)$. Thanks to Moshe Vardi for bringing their work to our attention, and thanks to Orna Grumberg for sending us a copy of their technical report.

[3] Direct simulation is called *strong* simulation in [EH00].

nondeterministic automaton. Thus, there is a need for efficient algorithms and heuristics that reduce the state space substantially.

We introduce a natural intermediate notion between direct and fair simulation, called *delayed* simulation, which satisfies property (**). We show that delayed simulation can yield substantially greater reduction, by an arbitrarily large factor, than direct simulation. We provide an algorithm for computing the entire delayed simulation relation which arises from precisely the same parity-game framework and has the same complexity as our algorithm for fair simulation.

Lastly, our parity game framework also easily accommodates computation of bisimulation relations (which are generally less coarse than simulation). In particular, we show that the fair bisimulation relation on Büchi automata can be computed in time $O(mn^3)$ and $O(mn)$ space. Fair bisimulation was studied by [HR00] for Büchi and Streett automata, who for Büchi automata gave an $O(n^{10})$ time algorithm to compute whether one state is fair bisimilar to another.

The paper is organized as follows: in Sect. 2, we define all (bi)simulation notions used in the paper. In Sect. 3 we show how for each simulation notion (and fair bisimulation), given a Büchi automaton, we can define a parity game that captures the (bi)simulation. In Sect. 4, we use our variant of Jurdzinski's algorithm for parity games to give efficient algorithms for computing several such (bi)simulation relations. In Sect. 5, we prove that the delayed simulation quotient can be used to reduce automaton size, and yields better reduction than direct simulation, but that the fair simulation quotient cannot be so used. We conclude in Sect. 6. Due to lack of space, most proofs must be omitted.

2 Delayed, Fair, and Other (Bi)Simulations

2.1 Simulation

We now define various notions of simulation, including fair and the new delayed simulation, in terms of appropriate games.[4] As usual, a Büchi automaton $A = \langle \Sigma, Q, q_I, \Delta, F \rangle$ has an alphabet Σ, a state set Q, an initial state $q_I \in Q$, a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, and a set of final states $F \subseteq Q$. We will henceforth assume that the automaton has no *dead ends*, i. e., from each state of A there is a path of length at least 1 to *some* state in F. It is easy to make sure this property holds without changing the accepting runs from any state, using a simple search to eliminate unnecessary states and transitions.

Recall that a *run* of A is a sequence $\pi = q_0 a_0 q_1 a_1 q_2 \ldots$ of states alternating with letters such that for all i, $(q_i, a_i, q_{i+1}) \in \Delta$. The ω-word associated with π is $w_\pi = a_0 a_1 a_2 \ldots$ The run π is *initial* if it starts with q_I; it is *accepting* if there exist infinitely many i with $q_i \in F$. The language defined by A is $L(A) = \{w_\pi \mid \pi \text{ is an initial, accepting run of } A\}$. We may want to change the start state of A to a different state q; the revised automaton is denoted by $A[q]$.

[4] For background on simulation, and its versions incorporating acceptance, see, e. g., [Mil89,HHK95], and [HKR97], respectively.

As in [HKR97], we define simulation game-theoretically.[5] The next definition presents all the notions of simulation we will consider: ordinary simulation, which ignores acceptance, as well as three variants which incorporate acceptance conditions of the given automaton, in particular, our new *delayed simulation*.

Definition 1. Given a Büchi automaton A and $(q_0, q_0') \in Q^2$, we define:

1. the *ordinary simulation game,* denoted $\boldsymbol{G}_A^o(q_0, q_0')$,
2. the *direct (strong) simulation game,* denoted $\boldsymbol{G}_A^{di}(q_0, q_0')$,
3. the *delayed simulation game,* denoted $\boldsymbol{G}_A^{de}(q_0, q_0')$,
4. the *fair simulation game,* denoted $\boldsymbol{G}_A^f(q_0, q_0')$.

Each of the games is played by two players, *Spoiler* and *Duplicator,* in rounds as follows. At the start, round 0, two pebbles, *Red* and *Blue*, are placed on q_0 and q_0', respectively. Assume that, at the beginning of round i, *Red* is on state q_i and *Blue* is on q_i'. Then:

1. Spoiler chooses a transition $(q_i, a, q_{i+1}) \in \Delta$ and moves *Red* to q_{i+1}.
2. Duplicator, responding, must chose a transition $(q_i', a, q_{i+1}') \in \Delta$ and moves *Blue* to q_{i+1}'. If no a-transition starting from q_i' exists, then the game halts and Spoiler wins.

Either the game halts, in which case Spoiler wins, or the game produces two infinite runs: $\pi = q_0 a_0 q_1 a_1 q_2 \ldots$ and $\pi' = q_0' a_0 q_1' a_1 q_2' \ldots$, built from the transitions taken by the two pebbles. Given these infinite runs, the following rules are used to determine the winner.

1. *Ordinary simulation:* Duplicator wins in any case. (In other words, fairness conditions are ignored; Duplicator wins as long as the game does not halt).
2. *Direct simulation:* Duplicator wins iff, for all i, if $q_i \in F$, then also $q_i' \in F$.
3. *Delayed simulation:* Duplicator wins iff, for all i, if $q_i \in F$, then there exists $j \geq i$ such that $q_j' \in F$.
4. *Fair simulation:* Duplicator wins iff there are infinitely many j such that $q_j' \in F$ or there are only finitely many i such that $q_i \in F$ (in other words, if there are infinitely many i such that $q_i \in F$, then there are also infinitely many j such that $q_j' \in F$).

Let $\star \in \{o, di, de, f\}$. A *strategy* for Duplicator in game $\boldsymbol{G}_A^\star(q_0, q_0')$ is a function $f\colon (\Sigma Q)^+ \to Q$ which, given the history of the game (actually, the choices of Spoiler) up to a certain point, determines the next move of Duplicator. Formally, f is a strategy for Duplicator if for every run $q_0 \rho a q$ of A, we have $(f(\rho), a, f(\rho a q)) \in \Delta$, where, by convention, $f(\epsilon) = q_0'$. A strategy f for Duplicator is a *winning* strategy if, no matter how Spoiler plays, Duplicator always wins.

[5] We will focus on simulations between distinct states of the same automaton ("autosimulations"), because we are primarily interested in state space reduction. Simulations between different automata can be treated by considering autosimulations between the states of the automaton consisting of their disjoint union. In [HKR97], the authors presented their work in terms of Kripke structures with fairness constraints. We use Büchi automata directly, where labels are on transitions instead of states. This difference is inconsequential for our results.

Formally, a strategy f for Duplicator is winning if whenever $\pi = q_0 a_0 q_1 a_1 \dots$ is an infinite run through A and $\pi' = q'_0 a_0 q'_1 a_1 q'_2 \dots$ is the run defined by $q'_{i+1} = f(a_0 q_1 a_1 q_2 \dots q_{i+1})$, then Duplicator wins based on π and π'.

Definition 2. Let A be a Büchi automaton. A state q' *ordinary, direct, delayed, fair*[6] *simulates* a state q if there is a winning strategy for Duplicator in $\boldsymbol{G}^{\star}_{A}(q, q')$ where $\star = o$, di, de, or f, respectively. We denote such a relationship by $q \preceq_{\star} q'$.

Proposition 1. *Let A be a Büchi automaton.*

1. *For $\star \in \{o, di, de, f\}$, $\preceq_{\star}$ is a reflexive, transitive relation (aka,* preorder *or* quasi-order*) on the state set Q.*
2. *The relations are ordered by containment: $\preceq_{di} \subseteq \preceq_{de} \subseteq \preceq_{f} \subseteq \preceq_{o}$.*
3. *For $\star \in \{di, de, f\}$, if $q \preceq_{\star} q'$, then $L(A[q]) \subseteq L(A[q'])$.*

Thus, delayed simulation is a new notion of intermediate "coarseness" between direct and fair simulation. We will see in Sect. 5 why it is more useful for state space reduction.

2.2 Bisimulation

For all the mentioned simulations there are corresponding notions of bisimulation, defined via a modification of the game. For lack of space, we will not provide detailed definitions for bisimulation. Instead we describe intuitively the simple needed modifications. The bisimulation game differs from the simulation game in that *Spoiler* gets to choose in each round which of the two pebbles, *Red* or *Blue*, to move and *Duplicator* has to respond with a move of the other pebble. The winner of the game is determined very similarly: if the game comes to a halt, *Spoiler* wins. If not, the winning condition for *Fair* bisimulation ([HR00]) is: "if an accept state appears infinitely often on one of the two runs π and π', then an accept state must appear infinitely often on the other as well". The winning condition for *Delayed* bisimulation is: "if an accept state is seen at position i of either run, then an accept state must be seen thereafter at some position $j \geq i$ of the other run". The winning conditions for *Direct* bisimulation becomes "if an accept state is seen at position i of either run, it must be seen at position i of both runs". Winning strategies for the bisimulation games are defined similarly. Bisimulations define an equivalence relation $\approx^{bi}_{\star}$ (not a preorder) on the state space, and the following containments hold: $\approx^{bi}_{di} \subseteq \approx^{bi}_{de} \subseteq \approx^{bi}_{f} \subseteq \approx^{bi}_{o}$. Generally, bisimulation is less coarse than the equivalence derived from the simulation preorder, which we describe in Sect. 5, i.e., $\approx^{bi}_{\star} \subseteq \approx_{\star}$.

[6] Our game definition of fair simulation deviates very slightly from that given in [HKR97], but is equivalent since we consider only automata with no dead ends.

3 Reformulating (Bi)Simulations as Parity Games

3.1 Simulation

We now show how, given a Büchi automaton A and $\star \in \{o, di, de, f\}$, we can obtain in a straightforward way a parity game graph $\boldsymbol{G}_A^\star$ such that the winning vertices in $\boldsymbol{G}_A^\star$ for Zero (aka Player 0) in the parity game determine precisely the pairs of states (q, q') of A where q' $\star$-simulates q. Importantly, the size of these parity game graphs will be $O(|Q||\Delta|)$, and the nodes of the game graphs will be labeled by at most three distinct "priorities". In fact, only one priority will suffice for $\boldsymbol{G}_A^o$ and $\boldsymbol{G}_A^{di}$, while $\boldsymbol{G}_A^{de}$ and $\boldsymbol{G}_A^f$ will use three priorities.

We briefly review here the basic formulation of a parity game. A parity game graph $\boldsymbol{G} = \langle V_0, V_1, E, p\rangle$ has two disjoint sets of vertices, V_0 and V_1, whose union is denoted V. There is an edge set $E \subseteq V \times V$, and $p\colon V \to \{0, \ldots, d-1\}$ is a mapping that assigns a *priority* to each vertex.

A parity game on $\boldsymbol{G}$, starting at vertex $v_0 \in V$, is denoted $\boldsymbol{P}(G, v_0)$, and is played by two players, *Zero* and *One*. The play starts by placing a pebble on vertex v_0. Thereafter, the pebble is moved according to the following rule: with the pebble currently on a vertex v_i, and $v_i \in V_0$ (V_1), Zero (One, respectively) plays and moves the pebble to a neighbor v_{i+1}, that is, such that $(v_i, v_{i+1}) \in E$.

If ever the above rule cannot be applied, i. e., someone can't move because there are no outgoing edges, the game ends, and the player who cannot move loses. Otherwise, the game goes on forever, and defines a path $\pi = v_0 v_1 v_2 \ldots$ in $\boldsymbol{G}$, called a *play* of the game. The winner of the game is then determined as follows. Let k_π be the minimum priority that occurs infinitely often in the play π, i. e., so that for infinitely many i, $p(v_i) = k_\pi$ and k_π is the least number with this property; Zero wins if k_π is even, whereas One wins if k_π is odd.

We now show how to build the game graphs $\boldsymbol{G}_A^\star$. All the game graphs are built following the same general pattern, with some minor alterations. We start with $\boldsymbol{G}_A^f$. The game graph $\boldsymbol{G}_A^f = \langle V_0^f, V_1^f, E_A^f, p_A^f\rangle$ will have only three priorities (i. e., the range of p_A^f will be $\{0, 1, 2\}$). For each pair of states $(q, q') \in Q^2$, there will be a vertex $v_{(q,q')} \in V_0^f$ such that Zero has a winning strategy from $v_{(q,q')}$ if and only if q' fair simulates q. Formally, $\boldsymbol{G}_A^f$ is defined by

$$V_0^f = \{v_{(q,q',a)} \mid q, q' \in Q \wedge \exists q''((q'', a, q) \in \Delta)\} \ , \tag{1}$$

$$V_1^f = \{v_{(q,q')} \mid q, q' \in Q\} \ , \tag{2}$$

$$E_A^f = \{(v_{(q_1,q_1',a)}, v_{(q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta\} \cup \{(v_{(q_1,q_1')}, v_{(q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta\} \ , \tag{3}$$

$$p_A^f(v) = \begin{cases} 0, & \text{if } (v = v_{(q,q',a)} \text{ or } v = v_{(q,q')}) \text{ and } q' \in F, \\ 1, & \text{if } v = v_{(q,q')},\ q \in F, \text{ and } q' \notin F, \\ 2, & \text{otherwise.} \end{cases} \tag{4}$$

We now describe how $\boldsymbol{G}_A^f$ can be modified to obtain $\boldsymbol{G}_A^o$ and $\boldsymbol{G}_A^{di}$, both of which require only trivial modification to $\boldsymbol{G}_A^f$. The parity game graph $\boldsymbol{G}_A^o$ is exactly

the same as $\boldsymbol{G}_A^f$, except that all nodes will receive priority 0, i.e., $p_A^o(v) = 0$ for all v. The parity game graph $\boldsymbol{G}_A^{di}$ is just like $\boldsymbol{G}_A^o$, meaning every vertex has priority 0, but some edges are eliminated:

$$E_A^{di} = E_A^f \setminus (\{(v, v_{(q_1,q_1')}) \mid q_1 \in F \wedge q_1' \notin F\} \cup \{(v_{(q_1,q_1')}, w) \mid q_1 \notin F \wedge q_1' \in F\}) \tag{5}$$

Finally, to define $\boldsymbol{G}_A^{de}$ we need to modify the game graph somewhat more. For each vertex of $\boldsymbol{G}_A^f$ there will be at most two corresponding vertices in $\boldsymbol{G}_A^{de}$:

$$V_0^{de} = \{v_{(b,q,q',a)} \mid q, q' \in Q \wedge b \in \{0,1\} \wedge \exists q''((q'', a, q) \in \Delta)\} \ , \tag{6}$$

$$V_1^{de} = \{v_{(b,q,q')} \mid q, q' \in Q \wedge b \in \{0,1\} \wedge (q' \in F \to b = 0)\} \ . \tag{7}$$

The extra bit b encodes whether or not, thus far in the simulation game, the *Red* pebble has witnessed an accept state without *Blue* having witnessed one since then. The edges of $\boldsymbol{G}_A^{de}$ are as follows:

$$\begin{aligned} E_A^{de} = \{&(v_{(b,q_1,q_1',a)}, v_{(b,q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta \wedge q_2' \notin F\} \\ &\cup \{(v_{(b,q_1,q_1',a)}, v_{(0,q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta \wedge q_2' \in F\} \\ &\cup \{(v_{(b,q_1,q_1')}, v_{(b,q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta \wedge q_2 \notin F\} \\ &\cup \{(v_{(b,q_1,q_1')}, v_{(1,q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta \wedge q_2 \in F\} \ . \end{aligned} \tag{8}$$

Lastly, we describe the priority function of $\boldsymbol{G}_A^{de}$:

$$p_A^{de}(v) = \begin{cases} b, & \text{if } v = v_{(b,q,q')}, \\ 2, & \text{if } v \in V_0. \end{cases} \tag{9}$$

In other words, we will assign priority 1 to only those vertices in V_1 that signify that an "unmatched" accept has been encountered by *Red*.[7]

The following lemma gathers a collection of facts we will need.

Lemma 1. *Let A be a Büchi automaton.*

1. *For $\star \in \{o, di, f\}$, Zero has a winning strategy in $\boldsymbol{P}(\boldsymbol{G}_A^\star, v_{(q_0,q_0')})$ if and only if q_0' $\star$-simulates q_0 in A. For $\star = de$, this statement holds if $v_{(q_0,q_0')}$ is replaced by $v_{(b,q_0,q_0')}$, letting $b = 1$ if $q_0 \in F$ and $q_0' \notin F$, and $b = 0$ otherwise.*
2. *For $\star \in \{o, di, de, f\}$, $|\boldsymbol{G}_A^\star| \in O(|\Delta||Q|)$.*
3. *For $\star \in \{f, de\}$, $|\{v \in V_A^\star \mid p_A^\star(v) = 1\}| \in O(|Q|^2)$.*

Since vertices of $\boldsymbol{G}_A^o$ and $\boldsymbol{G}_A^{di}$ only get assigned a single priority, we can dispense with algorithms for computing ordinary and direct simulation right away, matching the best known upper bounds: with one priority the winning set for Zero can be determined by a variant of AND/OR graph accessibility, computable in linear time (see, e. g., [And94]). Thus:

Corollary 1. *([HHK95, BP95]) Given a Büchi automaton A, with n states and m transitions, both $\preceq_o$ and $\preceq_{di}$ can be computed in time $O(mn)$.*

[7] Note that it is possible to use only two priorities in p_A^{de}, by assigning a vertex v the priority b, where b is the indicator bit of v. However, it turns out that using two priorities is a disadvantage over three because the encoding would not have property (3) of Lemma 1, which we need for our complexity bounds.

3.2 Bisimulation

$\star$-bisimulations can also be reformulated as parity games. For improving the complexity, such a reformulation only helps for fair bisimulation. Ordinary and direct bisimulation have known $O(m \log n)$ time algorithms ([PT87]), and we will see that delayed bisimulation corresponds to direct bisimulation after some linear time preprocessing on accept states of the Büchi automaton. We formulate fair bisimulation with a parity game graph G_A^{fbi} as follows. The vertices of G_A^{fbi} are

$$V_0^{fbi} = \{v_{(q,q',a,b_1,b_2)} \mid q, q' \in Q \wedge b_1, b_2 \in \{0,1\} \wedge \exists q''((q'', a, q) \in \Delta)\} \ , \quad (10)$$

$$V_1^{fbi} = \{v_{(q_0,q_1,b_2)} \mid q, q' \in Q \wedge b_2 \in \{0,1\}\} \ . \quad (11)$$

The two bits b_1 and b_2 will encode (1) which pebble was moved by *Spoiler* in this round, and (2) which of the two runs was latest to see (prior to this round) an accept state, respectively. For $q_0, q_1 \in Q$ and $b_2 \in \{0,1\}$, let

$$new(q_0, q_1, b_2) = \begin{cases} 0 & \text{if } q_0 \in F, \\ 1 & \text{if } q_0 \notin F \text{ and } q_1 \in F, \\ b_2 & \text{otherwise.} \end{cases}$$

The edges E_A^{fbi} are

$$\{(v_{(q_0,q_1,b_2)}, v_{(q_0',q_1',a,b_1,b_2')}) \mid (q_{b_1}, a, q_{b_1}') \in \Delta \wedge q_{(1-b_1)} = q_{(1-b_1)}' \wedge b_2' = new(q_0, q_1, b_2)\}$$
$$\cup \{(v_{(q_0,q_1,a,b_1,b_2)}, v_{(q_0',q_1',b_2)}) \mid (q_{(1-b_1)}, a, q_{(1-b_1)}') \in \Delta \wedge q_{b_1} = q_{b_1}'\} \quad (12)$$

The priority function is as follows. For $v \in V_0$, $p_A^{fbi}(v) = 2$, and for $v_{(q_0,q_1,b_2)} \in V_1$,

$$p_A^{fbi}(v_{(q_0,q_1,b_2)}) = \begin{cases} 0 & \text{if } q_{(1-b_2)} \in F, \\ 1 & \text{if } q_{(1-b_2)} \notin F \text{ and } q_{b_2} \in F, \\ 2 & \text{otherwise.} \end{cases} \quad (13)$$

The correspondence of this parity game and fair bisimulation is as follows.

Lemma 2. *Zero has a winning strategy in* $\boldsymbol{P}(\boldsymbol{G}_A^{fbi}, v_{(q_0,q_1,0)})$ *if and only if* q_0 *and* q_1 *are fair-bisimilar in* A*. Furthermore,* $|\boldsymbol{G}_A^{fbi}| \in O(|\Delta||Q|)$ *and* $|\{v \in V_A^{fbi} \mid p_A^{fbi}(v) = 1\}| \in O(|Q|^2)$*.*

To compute delayed bisimulation efficiently, we show that the delayed bisimulation relation corresponds to the direct bisimulation relation after some linear time preprocessing on the accept states of the Büchi automaton. Consider the following closure operation on accept states. Let $cl(A)$ be the Büchi automaton obtained from A by repeating the following until a fixed point is reached: while there is a state q such that all of its successors are in F, put q in F. Clearly, $cl(A)$ can be computed in linear time and $L(A) = L(cl(A))$.

Proposition 2. $q_1 \approx_{de}^{bi} q_2$ *in* A *if and only if* $q_1 \approx_{di}^{bi} q_2$ *in* $\mathrm{cl}(A)$*.*

Thus, $\approx_{de}^{bi}$ can also be computed in time $O(m \log n)$ ([PT87]).

4 Fast Algorithm for Computing Fair (Bi)Simulations and Delayed Simulations

We now use $\boldsymbol{G}_A^f$, $\boldsymbol{G}_A^{fbi}$, and $\boldsymbol{G}_A^{de}$ to give a fast algorithm for computing the relations $\preceq_f$, $\approx_f^{bi}$, and $\preceq_{de}$. Henceforth, we assume all parity game graphs have neither self loops nor dead ends. We can always obtain an "equivalent" such graph in linear time. To obtain the desired complexity bounds for solving parity games, we describe an efficient implementation of an algorithm by Jurdzinski [Jur00]. Jurdzinski uses progress measures (see also [Kla94,Wal96]) to compute the set of vertices in a parity game from which Zero has a winning strategy.

We start with some terminology. Let $\boldsymbol{G}$ be a parity game graph as before, n' its number of vertices, m' its number of edges, and assume there are only three priorities, that is, $p\colon V \to \{0,1,2\}$. Let $n_1 = |p^{-1}(1)|$. The algorithm assigns to each vertex a "progress measure" in the range $D = \{0,\ldots,n_1\} \cup \{\infty\}$. Initially, every vertex is assigned 0. The measures are repeatedly "incremented" in a certain fashion until a "fixed point" is reached. We assume D is totally ordered in the natural way. For $i < 3$ and $x \in D$, we define $\langle x\rangle_i$ as follows. First, $\langle x\rangle_0 = 0$ if $x < \infty$ and $\langle\infty\rangle_0 = \infty$. Second, $\langle x\rangle_1 = \langle x\rangle_2 = x$ for every $x \in D$. In the same spirit, $\mathit{incr}_0(x) = \mathit{incr}_2(x) = x$ for every x, and $\mathit{incr}_1(x) = x + 1$ where, by convention, $n_1 + 1 = \infty$ and $\infty + 1 = \infty$. For simplicity in notation, if $v \in V$, we write $\langle x\rangle_v$ and $\mathit{incr}_v(x)$ for $\langle x\rangle_{p(v)}$ and $\mathit{incr}_{p(v)}(x)$, respectively. For every function $\rho\colon V \to D$, called a *measure*, and $v \in V$, let

$$\mathit{val}(\rho, v) = \begin{cases} \langle \min(\{\rho(w) \mid (v,w) \in E\})\rangle_v, & \text{if } v \in V_0, \\ \langle \max(\{\rho(w) \mid (v,w) \in E\})\rangle_v, & \text{if } v \in V_1. \end{cases} \tag{14}$$

Jurdzinski defines a "lifting" operator, which, given a measure ρ and $v \in V$, gives a new measure. In order to define it, we first need to define how an individual vertex's measure is updated with respect to that of its neighbors:

$$\mathit{update}(\rho, v) = \mathit{incr}_v(\mathit{val}(\rho, v)) \ . \tag{15}$$

The "lifted" measure, $\mathit{lift}(\rho, v)\colon V \to D$, is then defined as follows:

$$\mathit{lift}(\rho, v)(u) = \begin{cases} \mathit{update}(\rho, v), & \text{if } u = v, \\ \rho(u), & \text{otherwise.} \end{cases} \tag{16}$$

Jurdzinski's algorithm is depicted in Figure 1.

```
1  foreach v ∈ V do ρ(v) := 0
2   while there exists a v such that  update(ρ, v) ≠ ρ(v) do
3      ρ := lift(ρ, v)
4   endwhile
```

Fig. 1. Jurdzinski's lifting algorithm

The outcome determines the winning set of vertices for each player as follows:

Theorem 1. *([Jur00]) Let $\boldsymbol{G}$ be a parity game with $p\colon V \to \{0,1,2\}$. Zero has a winning strategy from precisely the vertices v such that, after the lifting algorithm depicted in Fig. 1 halts, $\rho(v) < \infty$.*

Jurdzinski's algorithm needs at most $n'(n_1+1)$ iterations of the while loop. More precisely, Jurdzinski argues as follows. Each vertex can only be lifted n_1+1 times. A lifting operation at v can be performed in time $O(|\mathrm{Sucs}(v)|)$ where $\mathrm{Sucs}(v)$ denotes the set of successors of v. So, overall, he concludes, the running time is $O(m'n_1)$. In this analysis, it is implicitly assumed that one can, in constant time, decide if there is a vertex v such that $\mathit{update}(\rho, v) \neq \rho(v)$, and find such a vertex. We provide an implementation of Jurdzinski's algorithm that achieves this when the number of priorities d is bounded by a constant.

Our algorithm, depicted in Figure 2, maintains a set L of "pending" vertices v whose measure needs to be considered for lifting, because a successor has recently been updated resulting in a requirement to update $\rho(v)$. Further, we maintain arrays B and C that store, for each vertex v, the value $\mathit{val}(\rho, v)$ and the number of successors u of v with $\langle \rho(u) \rangle_{p(v)} = \mathit{val}(\rho, v)$, denoted $\mathit{cnt}(\rho, v)$.

```
 1  foreach v ∈ V do
 2    B(v) := 0; C(v) := |{w | (v,w) ∈ E}|; ρ(v) := 0;
 3  L := {v ∈ V | p(v) is odd};
 4  while L ≠ ∅ do
 5    let v ∈ L; L := L \ {v};
 6    t := ρ(v);
 7    B(v) := val(ρ,v); C(v) := cnt(ρ,v); ρ(v) := incr_v(B(v));
 8    P := {w ∈ V | (w,v) ∈ E};
 9    foreach w ∈ P such that w ∉ L do
10      if w ∈ V_0 and t = B(w) and C(w) > 1 then C(w) := C(w) − 1;
11      if w ∈ V_0 and t = B(w) and C(w) = 1 then L := L ∪ {w};
12      if w ∈ V_1 and ρ(v) = B(w) then C(w) := C(w) + 1;
13      if w ∈ V_1 and ρ(v) > B(w) then L := L ∪ {w};
14    endfor
15  endwhile
```

Fig. 2. Efficient implementation of the lifting algorithm

Lemma 3. *The lifting algorithm depicted in Fig. 2 computes the same function ρ as Jurdzinski's algorithm, in time $O(m'n_1)$ and space $O(m')$.*

Proof (sketch). Whether a vertex w needs to be placed on L is determined in constant time by maintaining, for each vertex w, the current "best measure" $B(w)$ of any of its successors, as well as the count $C(w)$ of how many such neighbors there are with the "best measure". The running time follows because each vertex can enter L at most n_1+1 times, and the time taken by the while loop body is proportional to the number of edges incident on the vertex. □

In the general case, where $p\colon V \to \{0,\ldots,d-1\}$, Jurdzinski's algorithm uses a more elaborate measure, where the elements of D are vectors of length $\lfloor d/2 \rfloor$. Our implementation of Jurdzinski's algorithm extends directly to the general case, yielding an additional factor d in comparison with the time bound claimed by Jurdzinski. We do not know how Jurdzinski's claimed bounds can be achieved in the general case without the extra factor d. Finally, using Lemmas 1, 2, and 3, we may conclude:

Theorem 2. *For a Büchi automaton A, $\preceq_f$, $\approx_f^{bi}$, and $\preceq_{de}$ can all be computed in time $O(|\Delta||Q|^3)$ and space $O(|Q||\Delta|)$.*

As mentioned, in prior work $O(|Q|^6)$-time ([HKR97]), and $O(|Q|^{10})$-time ([HR00]), algorithms were given for deciding whether $q \preceq_f q'$, and respectively $q \approx_f^{bi} q'$, hold for a *single* pair of states (q, q').

5 Reducing State Spaces by Quotienting: Delayed Simulation Is Better

In this section, we show: (1) quotienting with respect to delayed simulation preserves the recognized language, (2) that this is not true with fair simulation, and (3) quotients with respect to delayed simulation can indeed be substantially smaller than quotients with respect to direct simulation, even when the latter is computed on the "accept closure" $cl(A)$ (unlike what we saw with delayed bisimulation). We first define quotients.

Definition 3. For a Büchi automaton A, and an equivalence relation $\approx$ on the states of A, let $[q]$ denote the equivalence class of $q \in Q$ with respect to $\approx$. The *quotient* of A with respect to $\approx$ is the automaton $A/{\approx} = \langle \Sigma, Q/{\approx}, \Delta_{\approx}, [q_I], F/{\approx}\rangle$ where $\Delta_{\approx} = \{([q], a, [q']) \mid \exists\ q_0 \in [q],\ q_0' \in [q'],$ such that $(q_0, a, q_0') \in \Delta\}$.

In order to apply our simulation relations, we define, corresponding to each simulation preorder, an equivalence relation $\approx_o$, $\approx_{di}$, $\approx_{de}$, and $\approx_f$, where: $q \approx_\star q'$ iff $q \preceq_\star q'$ and $q' \preceq_\star q$. Note that both $\approx_\star$ and $A/{\approx_\star}$ can be computed from $\preceq_\star$ requiring no more time (asymptotically) than that to compute $\preceq_\star$ on A. The quotient with respect to $\approx_{di}$ preserves the language of any automaton, while this is obviously not true for $\approx_o$. We will later see that this is not true for $\approx_f$ either. We show that this is true for $\approx_{de}$.

Lemma 4. *Let A be a Büchi automaton.*

1. *If $q_0 \preceq_{de} q_0'$ and $(q_0, a, q_1) \in \Delta$, then there exists q_1' with $q_1 \preceq_{de} q_1'$ and $(q_0', a, q_1') \in \Delta$.*
2. *If $q_0 \preceq_{de} q_0'$ and $[q_0]_{de}\, a_0\, [q_1]_{de}\, a_1 \ldots$ is a finite or infinite run of $A/{\approx_{de}}$, then there exists a run $q_0' a_0 q_1' a_1 \ldots$ of A of the same length such that $q_i \preceq_{de} q_i'$ for every i.*
3. *If $q_0 \preceq_{de} q_0''$ and $[q_0]_{de}\, a_0\, [q_1]_{de}\, a_1 \ldots$ is an infinite run of $A/{\approx_{de}}$ with $q_0 \in F$, then there exists a finite run $q_0'' a_0 \ldots a_{r-1} q_r''$ of A such that $q_j \preceq_{de} q_j''$ for $j \leq r$ and $q_r'' \in F$.*

Theorem 3. *For any Büchi automaton A, $L(A) = L(A/{\approx_{de}})$.*

We can thus use $A/{\approx_{de}}$ to reduce the size of A, just as with direct simulation. In fact, $A/{\approx_{de}}$ can be smaller than $A/{\approx_{di}}$ (as well as $cl(A)/{\approx_{di}}$) by an arbitrarily large factor:

Proposition 3. *For $n \geq 2$, there is a Büchi automaton A_n with $n+1$ states such that $A_n/{\approx_{de}}$ has 2 states but $A_n/{\approx_{di}}$ has $n+1$ states (and $A_n = \mathrm{cl}(A_n)$).*

Proof. Consider automaton A_n in Figure 3. It is not hard to establish that in A_n each outer state delayed simulates each other outer state. Thus $A_n/{\approx_{de}}$ has 2 states. On the other hand, $A_n = cl(A_n)$, and no state of A_n direct simulates any other state of A_n. Thus $A_n/{\approx_{di}} = A_n$ and has $n+1$ states. □

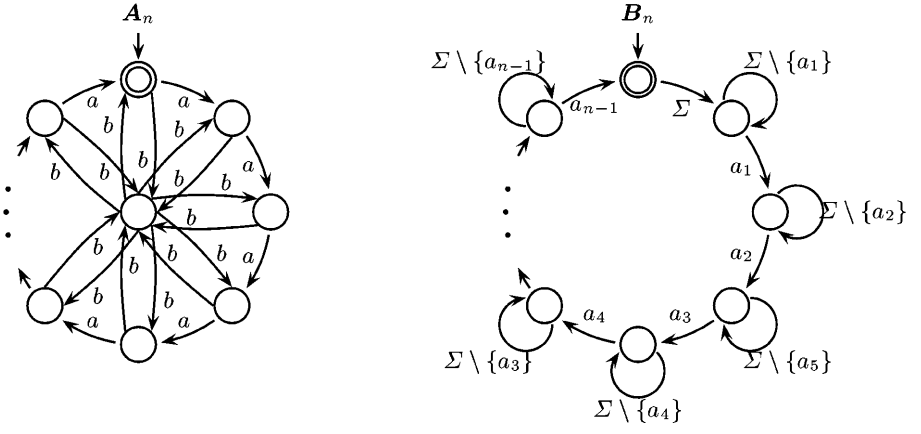


Fig. 3. Automata A_n and B_n

Next we see that Theorem 3 fails badly for fair simulation and bisimulation, that is, fair (bi)simulation cannot be used for state space reduction via quotienting. ([HR00] makes some related observations.)

Proposition 4. *For $n \geq 2$, there is a Büchi automaton B_n with n states, each of which fairly (bi)simulates every other state, but such that no Büchi automaton with fewer than n states accepts $L(B_n)$. In particular, $L(B_n) \neq L(B_n/{\approx_f^{bi}})$.*[8]

Proof. Consider the automaton B_n shown in Figure 3. It has n states, and an alphabet $\Sigma = \{a_1, \ldots, a_{n-1}\}$. To see that every state of B_n fair simulates (and fair bisimulates) every other state, first note that because the automaton is

[8] Note that this inequality holds under any "reasonable" definition of a quotient with respect to ${\approx_f^{bi}}$ or ${\approx_f}$, e.g., as long as states of the quotient are equivalence classes.

deterministic Duplicator has no choice in her strategy. A run (played by Spoiler) goes through the accept state infinitely often iff each a_i is encountered infinitely often. But this statement holds no matter which state the run begins from. Thus Duplicator's unique strategy from the initial state pair will be a winning strategy. The language $L(B_n)$ contains precisely those ω-words where each a_i occurs infinitely often. It is not hard to show that there are no Büchi automata recognizing $L(B_n)$ with fewer than n states. □

6 Conclusions

We have presented a unified parity game framework in which to understand optimal known algorithms for a variety of simulation notions for Büchi automata. In particular, we have improved upon the best bounds for fair simulation (and fair bisimulation), matched the best bound for ordinary simulation, and have presented an algorithm for the new notion of delayed simulation. Our algorithms employ a relatively simple fixed point computation, an enhancement of an algorithm by Jurdzinski for parity games, and should perform well in practice.

Our own main aim in using simulations is efficient state space reduction, as in [EH00]. We introduced delayed simulation and showed that, unlike fair simulation, delayed simulation quotients can be used for state space reduction, and allow greater reduction than direct (strong) simulation, which has been used in the past. Optimization of property automata prior to model checking is an important ingredient in making explicit state model checkers such as SPIN more efficient.

References

[And94] H. Andersen. Model checking and boolean graphs. *TCS*, 126(1):3–30, 1994.

[BG00] D. Bustan and O. Grumberg. Checking for fair simulation in models with Büchi fairness constraints, Dec. 2000. Tech. Rep. TR-CS-2000-13, Technion.

[BP95] B. Bloom and R. Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Science of Computer Programming*, 24(3):189–220, 1995.

[DHWT91] D. L. Dill, A. J. Hu, and H. Wong-Toi. Checking for language inclusion using simulation relations. In *Proceedings of CAV'91*, pages 329–341, 1991.

[EH00] K. Etessami and G. Holzmann. Optimizing Büchi automata. In *Proc. of 11th Int. Conf on Concurrency Theory (CONCUR)*, pages 153–167, 2000.

[GL94] O. Grumberg and D. Long. Model checking and modular verification. *ACM Trans. on Programming Languages and Systems*, 16(3):843–871, 1994.

[HHK95] M. Henzinger, T. Henzinger, and P. Kopke. Computing simulations on finite and infinite graphs. In *Proc. of 36th IEEE Symp. on Foundations of Comp. Sci. (FOCS'95)*, pages 453–462, 1995.

[HKR97] T. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *Proc. of 9th Int. Conf. on Concurrency Theory (CONCUR'97)*, number 1243 in LNCS, pages 273–287, 1997.

[HR00] T. Henzinger and S. Rajamani. Fair bisimulation. In *TACAS*, 2000.

[Jur00] M. Jurdziński. Small progress measures for solving parity games. In *STACS 2000, 17th Symp. on Theoretical Aspects of Computer Science*, volume 1770 of *LNCS*, pages 290–301. Springer-Verlag, 2000.

[Kla94] N. Klarlund. Progress measures,immediate determinacy, & a subset construction for tree automata. *Ann. Pure & Applied Logic*, 69:243–268, 1994.

[KV98] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. In *Proc. 30th ACM Symp. on Theory of Computing*, 1998.

[Mil89] R. Milner. *Communicatation and Concurrency*. Prentice-Hall, 1989.

[PT87] R. Paige and R. E. Tarjan. Three partition-refinement algorithms. *SIAM J. of Computing*, 16(6):973–989, 1987.

[SB00] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Proceedings of 12th Int. Conf. on Computer Aided Verification*, 2000.

[Wal96] I. Walukiewicz. Pushdown processes: games and model checking. In *Computer Aided Verification*, LNCS, pages 62–75. springer-verlag, 1996.

Hypergraphs in Model Checking: Acyclicity and Hypertree-Width versus Clique-Width*

Georg Gottlob[1] and Reinhard Pichler[2]

[1] Technische Universität Wien, gottlob@dbai.tuwien.ac.at
[2] Technische Universität Wien, reini@logic.at

Abstract. The principal aim of model checking is to provide efficient decision procedures for the evaluation of certain logical formulae over finite relational structures. Graphs and hypergraphs are important examples of such structures. If no restrictions are imposed on the logical formulae and on the structures under consideration, then this problem of model checking has a very high computational complexity. Hence, several restrictions have been proposed in the literature on the logical formulae and/or on the structures under consideration, in order to guarantee the tractability of this decision problem, e.g.: acyclicity, bounded tree-width, query-width and hypertree-width in case of queries as well as bounded tree-width and clique-width in case of structures. In this paper, we provide a detailed comparison of the expressive power of these restrictions.

1 Introduction

Model checking is the problem of deciding whether a logical formula or query Q is satisfied by a finite structure $\mathcal{S}$, which is formally written as $\mathcal{S} \models Q$. Q may be a formula in first-order logic, monadic second-order logic, existential second-order logic, and so on. Model checking is a central issue in database systems [1], where $\mathcal{S}$ represents a database and where the formula Q represents a database query. If Q is a closed formula, then Q is a Boolean query, otherwise $Q(\mathbf{x})$ with free variables $\mathbf{x}$ represents the query whose output consists of all tuples of domain values $\mathbf{a}$ such that $\mathcal{S} \models Q(\mathbf{a})$. Model checking is also a basic issue in the area of constraint satisfaction, which is essentially the same problem as conjunctive query evaluation [2,23]. Finally, model checking is used in computer aided verification [6], where $\mathcal{S}$ represents a state transition graph and Q is typically a formula of modal logic describing some temporal system behaviour. The results of the present paper are, however, more relevant to the former applications, namely, conjunctive database queries and constraint satisfaction.

Without any further restriction on the form of the structure and/or the query, these problems have a very high computational complexity. Hence, several restrictions have been proposed in the literature both for the structure and the query, in order to make these problems tractable. In particular, the evaluation

* This work was supported by the Austrian Science Fund (FWF) Project N.Z29-INF.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 708–719, 2001.

problem for classes of queries that are acyclic or have bounded tree-width, query-width or hypertree-width is tractable on arbitrary finite structures (combined complexity). On the other hand, arbitrary but fixed fixed formulae of monadic second order logic (MS_1) can be evaluated in polynomial time on graphs of bounded tree-width or bounded clique-width [10,13,12]. In other terms, MS_1 queries have polynomial data complexity on structures of bounded tree-width or of bounded query-width. MS_1 extends first order logic by the possibility of quantifying over monadic relational variables representing sets of vertices. Note that only the concept of bounded tree-width has so far been applied both to the queries and the structures. On the other hand, acyclicity, bounded query-width and hypertree-width have primarily been investigated as restrictions on the queries, while bounded clique-width has only been considered as a restriction on the structures. In this paper, we apply all of these restrictions both to the queries and to the structures. We shall thus answer the following questions:

- *Question 1*: How do the various notions of acyclicity and of bounded hypertree-width relate to the concept of bounded clique-width?

- *Question 2*: Are Boolean conjunctive queries of bounded clique-width tractable?

- *Question 3*: Bounded clique-width is currently the most general restriction on structures to make the model checking problem for monadic second order (MS_1) formulae tractable. Can the tractability barrier be pushed any further by using known generalizations of acyclicity that are more powerful than clique-width?

As for the first question, we provide an exact classification of the expressive power of the various restrictions. The result is depicted in Fig. 1. (Definitions of all of these concepts are provided in Sect. 2. Note that, throughout this paper, we only consider the clique-width of the *incidence* graph of a hypergraph and not of its *primal graph*. This point will be briefly discussed in Sect. 6). The arrows in the figure point from the less powerful concept to the more powerful one. All of the results relating clique-width with the various notions of acyclicity and query-width or hypertree-width are new, e.g.: It is shown that the class of γ-acyclic hypergraphs has bounded clique-width (of the incidence graph).

In [19] it was shown that the evaluation of Boolean conjunctive queries with bounded hypertree-width is tractable. Hence, together with our new results summarized in Fig. 1, we immediately get a positive answer to Question 2.

As for the third question, we prove that the restriction to hypergraphs of bounded query-width or of bounded β-hypertree-width is not sufficient to guarantee tractability of MS_1 queries. Thus bounded clique-width remains so far the most general restriction on structures that guarantees the tractability of arbitrary fixed MS_1 queries.

While tree-width can be recognized in linear time [4], it is currently unclear whether bounded clique-width can be recognized in polynomial time. We therefore propose *generalized tree-width (*gtw*)*, a cyclicity measure located between tree-width and clique width, and show that bounded gtw is recognizable in polynomial time.

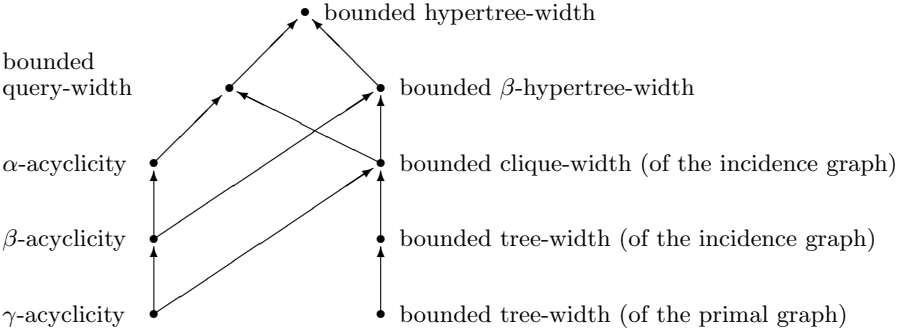


Fig. 1. Expressive power of various restrictions on hypergraphs

This paper is structured as follows: In Sect. 2 we recall some basic notions and results. Restrictions on the form of the queries and of the structures will be considered in the Sections 3 and 4, respectively. In Sect. 5, we introduce the notion of generalized tree-width. Finally, in Sect. 6, we give some concluding remarks. Due to space limitations, proofs can only be sketched or even have to be omitted. Full proofs of all the results presented here are given in [21].

2 Preliminaries

A *graph* is a pair $\langle \mathcal{V}, \mathcal{E} \rangle$ consisting of a set $\mathcal{V}$ of vertices (or nodes) and a set $\mathcal{E}$ of edges. We only consider undirected, loop-free graphs without multiple edges here. Hence, in particular, every edge $E \in \mathcal{E}$ is a two-element subset of $\mathcal{V}$. Two vertices are called *adjacent*, iff they are the endpoints of an edge in $\mathcal{E}$. A set $\mathcal{W} \subseteq \mathcal{V}$ is a *module* of a graph, iff the elements in $\mathcal{W}$ cannot be distinguished by the other vertices, i.e.: every vertex in $\mathcal{V} - \mathcal{W}$ is either adjacent to all vertices $W \in \mathcal{W}$ or to none. A *subgraph* is a pair $\langle \mathcal{V}', \mathcal{E}' \rangle$, s.t. $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$ hold. A subgraph is *induced*, iff $\mathcal{E}'$ is the restriction of $\mathcal{E}$ to those edges whose endpoints are in $\mathcal{V}'$. In a *labelled graph*, every vertex has exactly one label. A *k-graph* is a labelled graph where all labels are from the set $\{1, \ldots, k\}$, i.e.: A k-graph can be represented as $\langle \mathcal{V}, \mathcal{E}, \mathcal{V}_1, \ldots, \mathcal{V}_k \rangle$, where $\mathcal{V}$ is a set of vertices, $\mathcal{E}$ is a set of edges and the sets $\mathcal{V}_1, \ldots, \mathcal{V}_k$ are (possibly empty) subsets of $\mathcal{V}$ that form a partition of $\mathcal{V}$.

A *hypergraph* is a pair $\langle \mathcal{V}, \mathcal{H} \rangle$ consisting of a set $\mathcal{V}$ of vertices and a set $\mathcal{H}$ of hyperedges. A hyperedge $H \in \mathcal{H}$ is a subset of $\mathcal{V}$. A *subhypergraph* is a pair $\langle \mathcal{V}', \mathcal{H}' \rangle$, s.t. $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{H}' \subseteq \mathcal{H}$ hold. With every hypergraph $\mathcal{G} = \langle \mathcal{V}, \mathcal{H} \rangle$, we can associate the following two graphs: The *primal graph* (which is also called the *Gaifmann graph*) $\mathcal{P}(\mathcal{G})$ has the same vertices $\mathcal{V}$ as $\mathcal{G}$. Moreover, two vertices $V_1, V_2 \in \mathcal{V}$ are connected by an edge in $\mathcal{P}(\mathcal{G})$, iff there is a hyperedge $H \in \mathcal{H}$, s.t. both V_1 and V_2 are contained in $\mathcal{H}$. The *incidence graph* $\mathcal{I}(\mathcal{G})$ is a bipartite graph with vertices in $\mathcal{V} \cup \mathcal{H}$. Moreover, there is an edge in $\mathcal{I}(\mathcal{G})$ between two vertices $V \in \mathcal{V}$ and $H \in \mathcal{H}$, iff (in the hypergraph $\mathcal{G}$) V occurs in the hyperedge

H. The incidence graph can either be considered as an unlabelled graph or as a 2-graph, with the labels $\mathcal{V}$ and $\mathcal{H}$, respectively.

In order to determine the *clique-width* of a (labelled or unlabelled) graph, we have to deal with so-called *k-expressions* t and the graphs $G(t)$ generated by such k-expressions:

(1) Let $i \in \{1, \ldots, k\}$ and let V be a vertex. Then $i(V)$ is a k-expression. The corresponding graph consists of a single vertex V whose label is i.
(2) Let r and s be k-expressions that have no vertices in common. Then $r \oplus s$ is also a k-expression. The graph thus generated is the (disjoint) union of the graphs $G(r)$ and $G(s)$.
(3) Let $i, j \in \{1, \ldots, k\}$ with $i \neq j$ and let r be a k-expression. Then $\eta_{i,j}(r)$ is also a k-expression. The corresponding graph is the same as $G(r)$ augmented by edges from every vertex with label i to every vertex with label j.
(4) Let $i, j \in \{1, \ldots, k\}$ with $i \neq j$ and let r be a k-expression. Then $\rho_{i \to j}(r)$ is also a k-expression whose graph is the same as $G(r)$ except that all vertices with label i in $G(r)$ are relabelled to j.

The graph generated by a k-expression t can be either considered as a labelled graph with labels in $\{1, \ldots, k\}$ or as an unlabelled graph (by ignoring the labels assigned by t). Every subexpression s of a k-expression t generates a subgraph $G(s)$ of $G(t)$ (when considered as an unlabelled graph). The clique-width $\mathrm{cwd}(\mathcal{G})$ of a (labelled or unlabelled) graph $\mathcal{G}$ is the minimum k, s.t. there exists a k-expression that generates $\mathcal{G}$. Obviously, $\mathrm{cwd}(\mathcal{G}) \leq n$ for every graph with n vertices. It can be shown that every clique has clique-width 2, e.g. the clique with four nodes V_1, V_2, V_3, V_4 can be generated by the 2-expression $\eta_{1,2}(2(V_4) \oplus (\rho_{2 \to 1}(\eta_{1,2}(2(V_3) \oplus (\rho_{2 \to 1}(\eta_{1,2}(2(V_2) \oplus 1(V_1))))))))$.

A *tree decomposition* of a graph $\langle \mathcal{V}, \mathcal{E} \rangle$ is a pair $\langle T, \lambda \rangle$, where $T = \langle N, F \rangle$ is a tree and λ is a labelling function with $\lambda(p) \subseteq \mathcal{V}$ for every node $p \in N$, s.t. the following conditions hold:

(1) $\forall V \in \mathcal{V}$, $\exists p \in N$, s.t. $V \in \lambda(p)$.
(2) $\forall E \in \mathcal{E}$ with endpoints V_1 and V_2, $\exists p \in N$, s.t. $V_1 \in \lambda(p)$ and $V_2 \in \lambda(p)$.
(3) $\forall V \in \mathcal{V}$, the set $\{p \in N : V \in \lambda(p)\}$ induces a connected subtree of T.

The width of a tree decomposition $\langle T, \lambda \rangle$ is $\max(\{|\lambda(p)| - 1 : p \in N\})$. The tree-width $\mathrm{tw}(\mathcal{G})$ of a graph $\mathcal{G}$ is the minimum width over all its tree decompositions.

A *join tree* of a connected hypergraph $\langle \mathcal{V}, \mathcal{H} \rangle$ is a labelled tree $\langle T, \lambda \rangle$ with $T = \langle N, F \rangle$ and a labelling function λ with $\lambda(p) \in \mathcal{H}$ for every node $p \in N$. Moreover, the following conditions hold:

(1) $\forall H \in \mathcal{H}$, $\exists p \in N$, s.t. $\lambda(p) = H$.
(2) Let $\lambda(p_1) = H_1$ and $\lambda(p_2) = H_2$ for two distinct nodes p_1 and p_2. Moreover, let some vertex $V \in \mathcal{V}$ occur in both hyperedges H_1 and H_2. Then V must also occur in all hyperedges that are used as labels on the path from p_1 to p_2.

A hypergraph is α-acyclic, iff every connected component of this hypergraph has a join tree.

In [7], a *query decomposition* of a hypergraph $\langle \mathcal{V}, \mathcal{H} \rangle$ is defined as a pair $\langle T, \lambda \rangle$ where $T = \langle N, F \rangle$ is a tree and λ is a labelling function with $\lambda(p) \subseteq (\mathcal{V} \cup \mathcal{H})$ for every $p \in N$, s.t. the following conditions hold:

(1) $\forall H \in \mathcal{H}$, $\exists p \in N$, s.t. $H \in \lambda(p)$.
(2) $\forall V \in \mathcal{V}$, the set $\{p \in N : V \in \lambda(p)\} \cup \{q \in N : \exists H \in \mathcal{H}$, s.t. $H \in \lambda(q)$ and V occurs in the hyperedge $H\}$ induces a connected subtree of T.

Note that, in the original definition in [7], it is also required that $\forall H \in \mathcal{H}$, the set $\{p \in N : H \in \lambda(p)\}$ is a connected subtree of T. However, this condition is not needed for the evaluation of queries and it has therefore been omitted here. The width of a query decomposition is $\max(\{|\lambda(p)| : p \in N\})$. The query-width $\mathrm{qw}(\mathcal{G})$ of a hypergraph $\mathcal{G}$ is the minimum width of its query decompositions.

In [19], a *hypertree decomposition* of a hypergraph $\langle \mathcal{V}, \mathcal{H} \rangle$ is defined as a triple $\langle T, \chi, \lambda \rangle$ where $T = \langle N, F \rangle$ is a tree and χ and λ are labelling functions with $\chi(p) \subseteq \mathcal{V}$ and $\lambda(p) \subseteq \mathcal{H}$ for every $p \in N$, s.t. the following conditions hold:

(1) $\forall H \in \mathcal{H}$, $\exists p \in N$, s.t. $H \subseteq \chi(p)$, i.e.: "p covers H".
(2) $\forall V \in \mathcal{V}$, the set $\{p \in N : V \in \chi(p)\}$ induces a connected subtree of T.
(3) $\forall p \in N$, $\chi(p)$ contains only vertices that actually occur in at least one hyperedge of $\lambda(p)$.
(4) For every $p \in N$, if a vertex V occurs in some hyperedge $H \in \lambda(p)$ and if V is contained in $\chi(q)$ for some node q in the subtree below p, then V must also be contained in $\chi(p)$.

The width of a hypertree decomposition $\langle T, \chi, \lambda \rangle$ is $\max(\{|\lambda(p)| : p \in N\})$. The hypertree-width $\mathrm{hw}(\mathcal{G})$ of a hypergraph $\mathcal{G}$ is the minimum width over all its hypertree decompositions.

A conjunctive query Q is a first-order formula in prenex form whose only connectives are $\exists$ and $\wedge$. With every conjunctive query, we can associate a hypergraph $\mathcal{H}$, whose vertices $V_1, \ldots, V_n$ correspond to the variables $x_1, \ldots, x_n$ occurring in Q. Moreover, for every atom A with variables $\mathrm{Var}(A) = \{x_{i_1}, \ldots, x_{i_\alpha}\}$, there is a hyperedge $H = \{V_{i_1}, \ldots, V_{i_\alpha}\}$ in the hypergraph and vice versa. Then the notions of hypertree-width, query-width and acyclicity carry over in a natural way from hypergraphs to conjunctive queries. Likewise, the incidence graph or the primal graph of a conjunctive query Q is simply the corresponding graph of the associated hypergraph $\mathcal{H}$. Actually, the clique-width and the tree-width of a hypergraph or of a query can be either defined as the corresponding width of the incidence graph or the primal graph. If not indicated otherwise, we shall assume that the clique-width of a hypergraph or of a query refers to the incidence graph considered as an unlabelled graph.

Clique-width and tree-width are *hereditary* properties in that $\mathrm{cwd}(\mathcal{G}') \leq \mathrm{cwd}(\mathcal{G})$ and $\mathrm{tw}(\mathcal{G}') \leq \mathrm{tw}(\mathcal{G})$ hold for all induced subgraphs of a graph. In contrast, α-acyclicity, query-width and hypertree-width do not share this property, e.g.: a hypergraph $\mathcal{H}$ can be α-acyclic even though some subhypergraph $\mathcal{H}'$ is not. Likewise, $\mathcal{H}$ can have a subhypergraph $\mathcal{H}'$, s.t. $\mathrm{qw}(\mathcal{H}) < \mathrm{qw}(\mathcal{H}')$ or $\mathrm{hw}(\mathcal{H}) < \mathrm{hw}(\mathcal{H}')$ hold. The notions of *β-acyclicity* and *β-hypertree-width* can be regarded as the hereditary counterparts of α-acyclicity and hypertree-width: In [16], a hypergraph $\mathcal{H}$ is defined to be β-acyclic, iff every subhypergraph $\mathcal{H}'$ of $\mathcal{H}$ is α-acyclic. Analogously, we can define the β-hypertree-width of $\mathcal{H}$ as the $\max(\{hw(\mathcal{H}') : \mathcal{H}'$ is a subhypergraph of $\mathcal{H}\})$. In [16], another notion of acyclicity is presented, namely γ-acyclicity. Any hypergraph, that is γ-acyclic, is also β-acyclic. An algorithmic definition of γ-acyclicity will be given in Sect. 4.

3 Restricting the Form of the Queries

In this section we consider the case of arbitrary relational structures, where the form of the queries is subjected to some strong restrictions. Recall that the evaluation of arbitrary first-order queries is PSPACE-complete (cf. [24]). Actually, even if we restrict the form of first-order queries to conjunctive queries (where only conjunctions and existential quantification are allowed), then the query evaluation is still NP-complete (see [5]). If conjunctive queries are further restricted to α-acyclic conjunctive queries, then this problem becomes tractable (cf. [25]). However, acyclicity is a very severe restriction. Hence, in recent years, several attempts to deal with "almost acyclic queries" have been made. In particular, several notions of width have been introduced, in order to extend the class of tractable conjunctive queries, namely tree-width, query-width and hypertree-width (cf. [7], [17], [19], [22], [23]). In [19] and [20], it has been shown that, for some fixed k, the class of conjunctive queries with hypertree-width $\leq k$ properly contains the classes where the tree-width (of the incidence graph or of the primal graph) or the query-width, respectively, is bounded by k. Moreover, the concept of hypertree-width is a generalization of α-acyclicity in that a conjunctive query is acyclic, iff it has hypertree-width 1.

The notion of clique-width of (possibly labelled) graphs was originally introduced as a restriction on structures over which MS_1-formulae have to be evaluated. The restriction to bounded clique-width has proved to allow for a much larger class of structures than bounded tree-width, e.g.: In contrast to tree-width, a graph can have bounded clique-width, even if the degree of the vertices is unbounded. Moreover, bounded tree-width implies bounded clique-width, while the converse is in general not true (cf. [13]). In this section, we investigate the applicability of the criterion of bounded clique-width to (the incidence graph of) conjunctive queries. We show that conjunctive queries with bounded clique-width have bounded query-width and, therefore, also bounded hypertree-width. The converse can be easily shown to be not true (see [21]).

Theorem 3.1. (bounded clique-width implies bounded query-width) *Let Q be a conjunctive query whose incidence graph has clique-width $\leq k$. Then* $\mathrm{qw}(Q) \leq 2k$ *holds.*

Proof (Sketch). Let $\mathcal{H}$ be the hypergraph of a conjunctive query Q and let $\mathcal{I}$ be the incidence graph (considered as an unlabelled graph) of $\mathcal{H}$. Moreover, let s be a k-expression that generates $\mathcal{I}$. Recall that the nodes in $\mathcal{I}$ are divided into the nodes which correspond to a hyperedge and the nodes which correspond to a vertex in $\mathcal{H}$, respectively. Hence, the k-expression s can be transformed in the obvious way into a $2k$-expression t, s.t. t assigns each label $\ell \in \{1, \ldots, 2k\}$ either only to nodes corresponding to hyperedges or only to nodes corresponding to vertices in $\mathcal{H}$. Then we can show by structural induction how an appropriate query decomposition QD of $\mathcal{H}$ (or, equivalently, of the conjunctive query Q) can be constructed from t.

Note that every subexpression t' of t generates a subgraph $\mathcal{I}'$ of the incidence graph $\mathcal{I}$. Moreover, every such $\mathcal{I}'$ uniquely defines a hypergraph $\mathcal{H}'$. Now we can

construct a query decomposition QD' of $\mathcal{H}'$ in the following way (For details and for a proof of the correctness of this construction, see [21]):

Introduction of a new vertex: Let $t' = i(N)$ for some node N in $\mathcal{I}$, i.e.: N either corresponds to a hyperedge or to an H-vertex in $\mathcal{H}'$. The corresponding query decomposition QD' consists of a single node r whose label is the singleton $\{N\}$.

Disjoint union: Let $t' = s' \oplus s''$. Moreover, let $\mathcal{J}'$ and $\mathcal{J}''$ be the subgraphs of $\mathcal{I}$ defined by s' and s'', respectively. Finally, let $\mathcal{G}'$ and $\mathcal{G}''$ be the corresponding hypergraphs and let QD_1 and QD_2 be query decompositions of $\mathcal{G}'$ and $\mathcal{G}''$, respectively. Then we construct the query decomposition $QD' = \langle T', \lambda' \rangle$ in such a way that the tree T' has a new root node r and subtrees QD_1 and QD_2. Moreover, the roots r_1 and r_2 of QD_1 and QD_2, respectively, are appended to r as its child nodes.

The labelling λ of the nodes in QD_1 and QD_2 is left unchanged. As for the labelling $\lambda(r)$ of the new root r, we have to select an appropriate set of vertices and hyperedges of $\mathcal{G}'$ and $\mathcal{G}''$. This is done in the following way: Let $\mathcal{R}$ denote the set of all hyperedges or vertices in the hypergraphs $\mathcal{G}'$ and $\mathcal{G}''$, s.t. every $N \in \mathcal{R}$ either occurs in one of the labels of the roots r_1 and r_2 of QD_1 and QD_2, respectively, or N is a vertex from one of the hypergraphs $\mathcal{G}'$ and $\mathcal{G}''$, s.t. N occurs in some hyperedge H and this hyperedge H is contained in the label of either r_1 or r_2. By assumption, the hyperedges and vertices in $\mathcal{R}$ are assigned at most $2k$ different labels by t'. Then we get the desired set $\lambda(r)$ by selecting one representative for each label according to t'.

Introduction of edges: Let $t' = \eta_{i,j}(s)$ and let $QD_1 = \langle T_1, \lambda_1 \rangle$ be the query decomposition corresponding to s. Then the desired query decomposition $QD' = \langle T', \lambda' \rangle$ of $\mathcal{H}'$ is obtained in the following way: The tree T' consists of a new root node r and the subtree T_1, s.t. the root r_1 of T_1 is the only child of r. The labelling function λ' is defined as follows: The labelling of the root r_1 of T_1 is left unchanged, i.e.: $\lambda'(r_1) = \lambda_1(r_1)$. It can be shown that $\lambda'(r_1)$ contains at least one hyperedge G, s.t. the label of G (when considered as a node in the incidence graph $\mathcal{I}'$) according to t' is j. Then, for every node $p \neq r_1$ in T_1, s.t. $\lambda_1(p)$ contains a vertex with label i according to s, we replace these vertices in $\lambda_1(p)$ by the hyperedge G. For all other nodes p in T_1, we set $\lambda'(p) = \lambda_1(p)$. The labelling $\lambda'(r)$ of the new root r is constructed analogously to the case of the disjoint union discussed above, i.e.: Let $\mathcal{R}$ denote the set of all hyperedges or vertices in the hypergraph $\mathcal{G}$ corresponding to s, s.t. every $N \in \mathcal{R}$ either occurs in the label of the root r_1 of QD_1 or N is a vertex in the hypergraph $\mathcal{H}'$, s.t. N occurs in some hyperedge H in $\mathcal{H}'$ and this hyperedge H is contained in the label of r_1. By assumption, the hyperedges and vertices in $\mathcal{R}$ have at most $2k$ different labels according to t'. Then we select one representative for each label according to t' in order to get the desired set $\lambda'(r)$ of hyperedges and vertices.

Renaming of labels: Let $t' = \rho_{i \to j}(s)$. Moreover, let QD_1 be the query decomposition corresponding to s. It can be shown that then QD_1 is also the desired query decomposition corresponding to t'. □

Recall from [19] that $\mathrm{qw}(\mathcal{H}) \geq \mathrm{hw}(\mathcal{H})$ holds for every hypergraph $\mathcal{H}$. Hence, bounded clique-width clearly also implies bounded hypertree-width. Note that

this result has another interesting aspect: Apart from the special case of $k \leq 3$, it is not known whether graphs with clique-width $\leq k$ can be recognized in polynomial time for fixed k (cf. [8]). In contrast, conjunctive queries (or, equivalently, hypergraphs) with hypertree-width $\leq k$ actually can be recognized in polynomial time (cf. [19]). Hence, apart from being the more general concept, bounded hypertree-width also has better properties as far as recognizing such conjunctive queries is concerned.

4 Restricting the Form of the Structures

In [9], the complexity of testing certain graph properties is investigated. In terms of model checking, this corresponds to evaluating a fixed query over finite graphs. If the queries are restricted to (arbitrary but fixed) first-order formulae, then this problem is tractable for all finite graphs without any further restrictions. However, the expressive power of first-order logic is comparatively weak. Hence, attempts were made to investigate larger classes of queries. In particular, it was shown that the evaluation of fixed MS_1-formulae over graphs remains tractable, if we require the graphs to have bounded tree-width or bounded clique-width (cf. [9], [10], [11]).

In this section we consider monadic second-order queries over hypergraphs, where quantification is allowed over variables that stand for (sets of) vertices or hyperedges. Moreover, there are two unary predicates P_V, P_H and a binary predicate edg with the following meaning: $P_V(x)$, $P_H(x)$ state that the argument x is a vertex or a hyperedge, respectively, of the hypergraph. By $edg(v, h)$ we can express that the vertex v is contained in the hyperedge h. Clearly, these formulae correspond to MS_1-formulae that are evaluated over the incidence graphs (when considered as a labelled graph with two labels) of hypergraphs. Thus, the evaluation of such formulae is tractable, if the incidence graphs under consideration have bounded clique-width. From [10] we know that a class of labelled graphs with p labels (for fixed $p \geq 1$) has bounded clique-width, iff the same graphs without labels have bounded clique-width. Hence, in the sequel, we shall ignore the two different labels of the nodes of the incidence graph, since they have no effect on the tractability of the evaluation of MS_1-formulae.

It has already been mentioned that clique-width is a hereditary property while α-acyclicity and hypertree-width are not. In case of restrictions on the queries, this does not matter. However, if we look for appropriate restrictions on the structures, then it can be easily verified that α-acyclicity or bounded hypertree-width will clearly not suffice to make the evaluation of any fixed MS_1-formula tractable. Instead, we shall consider β-acyclicity and β-hypertree-width here as well as γ-acyclicity, which is even slightly more restrictive than β-acyclicity.

In [14], D'Atri and Moscarini provided an algorithm for recognizing γ-acyclic hypergraphs. (For details, see the original paper or [16]). In terms of the incidence graph of a hypergraph, we get an algorithm $\mathcal{A}$ consisting of the following rules:

1. *deletion of isolated nodes*: If a node N in $\mathcal{I}$ has no adjacent node, then N may be deleted.

2. *deletion of "ear nodes"*: If a node N in $\mathcal{I}$ has exactly one adjacent node, then N may be deleted.
3. *contraction of two-element modules*: If two vertices N and N' in $\mathcal{I}$ are adjacent to exactly the same nodes, then one of them may be deleted. We require that this rule may only be applied, if at least one such adjacent node exists.

A hypergraph $\mathcal{H}$ is γ-acyclic, iff the exhaustive, non-deterministic application of the above rules transforms the incidence graph $\mathcal{I}$ of $\mathcal{H}$ into the empty graph. The following theorem states that the class of γ-acyclic hypergraphs provides a lower bound on the class of hypergraphs with bounded clique-width. Due to space limitations, we can only state the result without a proof here. A proof is given in [21].

Theorem 4.1. (γ-acyclicity implies bounded clique-width) *The incidence graphs of γ-acyclic hypergraphs have clique-width ≤ 3.*

An upper bound on the class of hypergraphs with bounded clique-width can be obtained by comparing bounded clique-width with β-acyclicity and β-hypertree-width, respectively.

Theorem 4.2. (clique-width versus β-acyclicity) *The class of incidence graphs of β-acyclic hypergraphs has unbounded clique-width.*

Proof (Sketch). Consider the sequence $(\mathcal{H}_n)_{n\geq 1}$ of hypergraphs, where $\mathcal{H}_n$ has the vertices $V = \{y_1, \ldots, y_n\} \cup \{x_{ij} : 1 \leq i < j \leq n\}$ and the n hyperedges $H_1, \ldots, H_n$ with $H_l = \{y_l\} \cup \{x_{\alpha\beta} : \alpha < \beta \leq l\} \cup \{x_{l\gamma} : l < \gamma \leq n\}$, i.e. $H_1 = \{y_1, x_{12}, x_{13}, \ldots, x_{1n}\}$, $H_2 = \{y_2, x_{12}, x_{23}, \ldots, x_{2n}\}$, $H_3 = \{y_3, x_{12}, x_{13}, x_{23}, x_{34}, \ldots, x_{3n}\}$, $\ldots$, $H_n = \{y_n, x_{12}, \ldots, x_{1n}, x_{23}, \ldots, x_{2n}, \ldots, x_{(n-1)n}\}$. Then it can be shown that, on the one hand, $\mathcal{H}_n$ is β-acyclic and, on the other hand, the incidence graph $\mathcal{I}_n$ of $\mathcal{H}_n$ has clique-width $\geq n$. □

Recall from Theorem 3.1 that if the clique-width of the incidence graph of a hypergraph $\mathcal{H}$ is $\leq k$, then we have $\mathrm{qw}(\mathcal{H}) \leq 2k$ and hence also $\mathrm{hw}(\mathcal{H}) \leq 2k$. Actually, an inspection of the proof of Theorem 3.1 reveals that this construction of a query decomposition also works for every subhypergraph $\mathcal{H}'$ of $\mathcal{H}$. In other words, we get the following result:

Theorem 4.3. (bounded clique-width implies bounded β-hypertree-width) *Let $\mathcal{H}$ be a hypergraph whose incidence graph has clique-width $\leq k$. Then $\mathcal{H}$ has β-hypertree-width $\leq 2k$.*

By the Theorems 4.2 and 4.3 we know that the class of hypergraphs with bounded β-hypertree-width strictly extends the class of hypergraphs whose incidence graph has bounded clique-width. Now the question naturally arises as to whether bounded β-hypertree-width of the structures under consideration suffices to guarantee the tractability of the evaluation of any MS_1-formula. Unfortunately, the answer given in Theorem 4.4 below is negative. Thus, bounded clique-width remains the concept with the highest expressive power known so far, s.t. the evaluation of MS_1-queries is still tractable (cf. Fig. 1).

Theorem 4.4. (MS_1-queries and bounded β-hypertree-width) *The evaluation of an arbitrary fixed MS_1-query over hypergraphs with bounded β-hypertree-width is, in general, not tractable, i.e.: There exist NP-complete problems that can be encoded as MS_1-queries on such hypergraphs.*

Proof (Sketch). Let $\mathcal{G} = \langle V, E\rangle$ be an arbitrary graph and let $\mathcal{H} = \langle V, H\rangle$ be a hypergraph, where the set H of hyperedges is defined as follows: $H = \{V - \{x, y\} : \{x, y\}$ is an edge in $E\}$, i.e.: every edge e of $\mathcal{G}$ is encoded by a hyperedge which contains all vertices from V except for the endpoints of e. Then the hypergraph $\mathcal{H}$ can be shown to have β-hypertree-width ≤ 3. Moreover, the well-known NP-complete graph-3-colourability problem can be expressed as an MS_1-query of the form $(\exists C_1)(\exists C_2)(\exists C_3)$ "C_1, C_2 and C_3 provide a partition of V" $\wedge$ $(\forall x)(\forall y)[\, P_V(x) \wedge P_V(y) \wedge (\exists h)(P_H(h) \wedge \neg edg(x, h) \wedge \neg edg(y, h)) \rightarrow$ "x and y have different colours"]. □

5 Generalized Tree-Width

As has already been mentioned in Sect. 2, clique-width is much more powerful than tree-width. On the other hand, the lack of an efficient procedure for recognizing graphs with clique-width $\leq k$ for some arbitrary but fixed k is a major drawback of clique-width. Hence, it is worth trying to extend the notion of tree-width to some kind of "generalized tree-width", which is more powerful than tree-width and which is still efficiently recognizable. One such generalization is proposed below.

Recall from [15], that the existence of a big complete bipartite graph as a subgraph of a graph $\mathcal{G}$ has a very bad effect on the tree-width of $\mathcal{G}$, e.g.: consider the sequence $(\mathcal{H}_n)_{n\geq 1}$ of hypergraphs, where $\mathcal{H}_n$ has vertices $V = \{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_n\}$ and n hyperedges $H_1, \ldots, H_n$ with $H_i = \{y_i, x_1, \ldots, x_n\}$. Then, for every n, the (incidence graph $\mathcal{I}_n$ of the) hypergraph $\mathcal{H}_n$ has tree-width n, since it contains the complete bipartite graph with nodes $\{x_1, \ldots, x_n\}$ and $\{H_1, \ldots, H_n\}$, respectively. On the other hand, for every n, $\mathcal{H}_n$ is γ-acyclic, i.e.: The simple transformations of the algorithm $\mathcal{A}$ from Sect. 4 suffice to reduce the incidence graph of $\mathcal{H}_n$ to the empty graph. In particular, the complete bipartite graphs contained in the incidence graphs of these hypergraphs can be eliminated by these simple transformations. It therefore makes sense to consider the following generalization of the tree-width:

Definition 5.1. (generalized tree-width) *Let $\mathcal{G}$ be an arbitrary graph and let $\mathcal{G}'$ be the graph that results from exhaustive application of the following rules: deletion of isolated nodes, deletion of ear nodes and contraction of two-element modules. Then we define the* generalized tree-width *of $\mathcal{G}$ as* $\mathrm{gtw}(\mathcal{G}) = \mathrm{tw}(\mathcal{G}')$.

It is not hard to show that $\mathrm{gtw}(\mathcal{G})$ is well-defined. Moreover, a polynomial time algorithm for recognizing the graphs with gtw $\leq k$ for some fixed k can be constructed in the obvious way, namely: First, an input graph $\mathcal{G}$ is transformed into $\mathcal{G}'$ via the transformation from Definition 5.1 above. Then we can apply the algorithm from [3], which decides in linear time, whether $\mathrm{tw}(\mathcal{G}') \leq k$

holds. Moreover, recall from [13], that for every undirected graph $\mathcal{G}$, the relation $\mathrm{cwd}(\mathcal{G}) \leq 2^{\mathrm{tw}(\mathcal{G})+1} + 1$ holds. By combining this result with the proof idea of Theorem 4.1, it can be shown that bounded generalized tree-width implies bounded clique-width and, therefore, bounded generalized tree-width suffices to guarantee the tractability of the evaluation of MS_1-queries. Of course, the class of graphs with bounded generalized tree-width is smaller than the class of graphs with bounded clique-width. However, the advantage of this new concept is that, in contrast to bounded clique-width, it can be efficiently recognized.

6 Conclusion

In this paper, we have compared several notions of acyclicity and hypertree-width of hypergraphs on the one hand with clique-width (of the corresponding incidence graphs) on the other hand. This comparison has been done both if these restrictions are imposed on the queries and on the structures. Note that we have only considered the clique-width of the incidence graph here. When considering restrictions on conjunctive queries, this choice is somehow justified. In particular, it can be easily shown that there are NP-hard classes of queries whose primal graphs have bounded clique-width (cf. [21]). However, when considering restrictions on the form of the structures, the primal graphs also play an important role. Actually, it can be shown that β-acyclicity and bounded clique-width of the primal graph are uncomparable (cf. [21]). However, the exact position of bounded clique-width of the primal graph in Fig. 1 has to be determined yet.

In Sect. 5 we have shown how the insights from the comparison of γ-acyclicity with bounded clique-width can be used for an easy generalization of the tree-width. As long as no polynomial time algorithm for recognizing graphs with clique-width $\leq k$ (for some arbitrary but fixed k) has been found, the search for an appropriate generalization of the tree-width is an interesting research area. We have provided a first and very simple step in this direction, to which further steps should be added.

References

1. S.Abiteboul, R.Hull, V.Vianu. *Foundations of Databases*, Addison-Wesley Publishing Company (1995).
2. W.Bibel. Constraint Satisfaction from a Deductive Viewpoint. In *Artificial Intelligence*, Vol 35, pp. 401-413 (1988).
3. H.L.Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. In *SIAM Journal on Computing*, Vol 25, No 6, pp. 1305-1317 (1996).
4. H.L.Bodlaender. Treewidth: Algorithmic Techniques and Results. In *Proc. of MFCS'97*, LNCS 1295, pp. 19-36, Springer, (1997).
5. A.K.Chandra, P.M.Merlin. Optimal Implementation of Conjunctive Queries in Relational Databases. In *Proc. of STOC'77*, pp. 77-90, ACM Press (1977).
6. E.M.Clarke, O.Grumberg, D.Peled. *Model Checking*, MIT Press (1999).
7. Ch.Chekuri, A.Rajaraman. Conjunctive Query Containment Revisited. In *Proc. of ICDT'97*, LNCS 1186, pp. 130-144, Springer (1997).

8. D.G.Corneil, M.Habib, J.-M.Langlinel, B.Reed, U.Rotics. Polynomial Time Recognition of clique-width ≤ 3 graphs, extended abstract. In *Proc. of LATIN 2000*, LNCS 1776, pp. 126-134, Springer (2000).
9. B.Courcelle. Graph Rewriting: An Algebraic and Logic Approach. In *Handbook of Theoretical Computer Science*, Vol 2, pp. 194-241, J. van Leeuwen (ed.), Elsevier Science Publishers (1990).
10. B.Courcelle. Monadic second-order logic of graphs VII: Graphs as relational structures. In *Theoretical Computer Science*, Vol 101, pp. 3-33 (1992).
11. B.Courcelle, J.Engelfriet, G.Rozenberg. Handle-rewriting hypergraph grammars. In *Journal of Computer and System Sciences*, Vol 46, pp. 218-270 (1993).
12. B.Courcelle, J.A.Makowsky, U.Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. In *Theory of Computing Systems*, Vol 33, pp. 125-150 (2000).
13. B.Courcelle, S.Olariu. Upper bounds on the clique-width of graphs. In *Discrete Applied Mathematics*, Vol 101, pp. 77-114 (2000).
14. A.D'Atri, M.Moscarini. Acyclic Hypergraphs: Their recognition and top-down versus bottom-up generation. Technical Report R.29, Consiglio Nazionale delle Ricerche, Istituto di Analisi dei Sistemi ed Informatica (1982).
15. R.G.Downey, M.R.Fellows. *Parameterized Complexity*, Springer-Verlag (1997).
16. R.Fagin. Degrees of Acyclicity for Hypergraphs and Relational Database Schemes. In *Journal of the ACM*, Vol 30, No 3, pp. 514-550 (1983).
17. J. Flum, M. Frick, and M. Grohe. Query Evaluation via Tree-Decomposition. In *Proc. of ICDT'01*. Currently available at `www.math.uic.edu/grohe/pub/query.ps`
18. G.Gottlob, N.Leone, F.Scarcello. The Complexity of Acyclic Conjunctive Queries. In *Proc. of FOCS'98*, pp. 706-715, (1998). Full paper to appear in JACM.
19. G.Gottlob, N.Leone, F.Scarcello. Hypertree Decompositions and Tractable Queries. In *Proc. of PODS'99*, pp. 21-32, ACM Press (1999).
20. G.Gottlob, N.Leone, F.Scarcello. A Comparison of Structural CSP Decomposition Methods. In *Proc. of IJCAI'99*, pp. 394-399, Morgan Kaufmann, (1999).
21. G.Gottlob, R.Pichler. Hypergraphs in Model Checking: Acyclicity and Hypertree-Width versus Clique-Width. Full paper. Available from the authors (2001).
22. M.Grohe, T.Schwentick, and L.Segoufin. When is the Evaluation of Conjunctive Queries Tractable? Manuscript, currently available at: `www.math.uic.edu/grohe/pub/grid.ps` (2001).
23. Ph.G.Kolaitis and M.Y.Vardi. Conjunctive-Query Containment and Constraint Satisfaction. In *Proc. of PODS'98*, pp. 205-213, ACM Press (1998).
24. K.Kunen: Answer Sets and Negation as Failure. In *Proc. of the Fourth Int. Conf. on Logic Programming*, Melbourne, pp. 219-228 (1987).
25. M. Yannakakis. Algorithms for Acyclic Database Schemes. In *Proc. of Int. Conf. on Very Large Data Bases (VLDB'81)*, pp. 82-94, Cannes, France (1981).

From Finite State Communication Protocols to High-Level Message Sequence Charts

Anca Muscholl[1] and Doron Peled[2]

[1] LIAFA, Université Paris VII
2, pl. Jussieu, case 7014
F-75251 Paris cedex 05
[2] Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974, USA

Abstract. The ITU standard for MSCs provides a useful framework for visualizing communication protocols. HMSCs can describe a collection of MSC scenarios in early stages of system design. They extend finite state systems by partial order semantics and asynchronous, unbounded message exchange.
Usually we ask whether an HMSC can be implemented, for instance by a finite state protocol. This question has been shown to be undecidable [5]. Motivated by the paradigm of reverse engineering we study in this paper the converse translation, specifically the question whether a finite state communication protocol can be transformed into an equivalent HMSC. This kind of translation is needed when e.g. different forms of specification (HMSC, finite automata, temporal logic) must be integrated into a single one, for instance into an HMSC.
We show in this paper that translating finite state automata into HMSCs is feasible under certain natural assumptions. Specifically, we show that we can test in polynomial time whether a finite state protocol given by a Büchi automaton is equivalent to an HMSC, provided that the automaton satisfies the diamond property (the precise bound is NLOGSPACE-complete). The diamond property is a natural property induced by concurrency. Under the weaker assumption of bounded Büchi automata we show that the test is co-NP-complete. Finally, without any buffer restriction the problem is shown to be undecidable.

Keywords: *Message sequence charts, specification, HMSC, bounded automata, partial order specification.*

1 Introduction

Message Sequence Charts (MSC) is a popular formalism used in software development. As other ITU standards, MSC also has a visual notation and therefore it is more appealing than textual formalisms. MSCs are often used in the early design of communication protocols for capturing requirements of a system. They have been known for a long time as sequence or timing diagrams. A similar formalism exists in UML for describing interactions between objects (object diagrams).

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 720–731, 2001.

High-level MSCs (HMSC) can specify collections of MSCs. The standard description of the ITU'96 norm consists of a graph where each node contains one MSC. Each maximal path starting from a designated initial state corresponds to a single execution (scenario). Such an execution can be used to denote the communication structure of a typical or an exceptional behavior of a system, or a counterexample found during testing or model checking.

MSCs are based on partial order semantics and do not impose any limitation on buffers (except for fifo message exchange). Therefore, HMSCs are infinite state systems and some verification problems such as model-checking [2] or race conditions [7] are undecidable. One possible solution is to restrict HMSCs in a syntactic way in order to get a finite state space. To this purpose, bounded HMSCs have been proposed in [2,7]. However, because of their partial order semantics automatic verification is expensive in this case. For instance, positive model-checking in form of inclusion of bounded HMSCs is EXPSPACE-complete [7].

Using HMSCs for partial specification of system requirements means that they are used in a specification process of stepwise refinement. At the implementation stage one important issue is whether an HMSC can be realized, for example by a finite state machine. However, this question turns out to be undecidable [5]. Motivated by the paradigm of reverse engineering we study in this paper the converse translation, specifically the question whether a finite state communication protocol can be transformed into an equivalent HMSC. This form of translation is also required when one integrates different kinds of specification (HMSC, finite automata, temporal logic) into a single one. Since HMSC is a standardized, graphical notation, it is reasonable to provide an HMSC specification, whenever possible.

The translation from finite automata to HMSCs is well justified by the computational results obtained in this paper. We show that translating finite automata into HMSCs is feasible under certain natural assumptions. Specifically, we show that we can test in polynomial time whether a finite state protocol given by a Büchi automaton is equivalent to an HMSC, provided that the automaton satisfies the diamond property (the precise bound is NLOGSPACE-complete). The diamond property is a natural property in concurrent systems. Under the weaker assumption of bounded Büchi automata we show that equivalence to HMSC is co-NP-complete. The advantage of using bounded Büchi automata is that one is allowed to specify only representatives of MSC scenarios, thus the specification is more compact. More precisely, a bounded automaton might not accept an MSC language, however its MSC closure is still a regular language. Finally, without any buffer restriction the problem is shown to be undecidable.

Related work. Our results extend the recent decidability results obtained for DFA and languages of finite MSCs in [5]. A complementary approach are Compositional MSCs, which were proposed in [3] for representing every finite state protocol as an MSC-like graph, by allowing communication between sequentially connected nodes. However, this extension has to cope with undecidability of simple properties, similar to communicating finite state machines.

2 Definitions and Notations

In this section we define message sequence charts (MSC) and high-level message sequence charts (HMSC), based on the ITU'96 standard. Each MSC describes a scenario or an execution, where processes (instances) communicate with each other. An MSC scenario includes a description of the messages sent, messages received, the local events, and the ordering between them. Ordering of events is enforced by the instance ordering and the message ordering. In the visual description of MSCs, each instance is represented by a vertical line, which gives a total order on the events belonging to that instance. Messages are usually represented by horizontal or slanted arrows from the sending process to the receiving one, see Figure 1.

Throughout the paper we consider finite or infinite MSCs over a set of processes (instances) $\mathcal{P} = \{P1, \ldots, Pn\}$ and we denote by $[n]$ the set of process indexes $\{1, \ldots, n\}$.

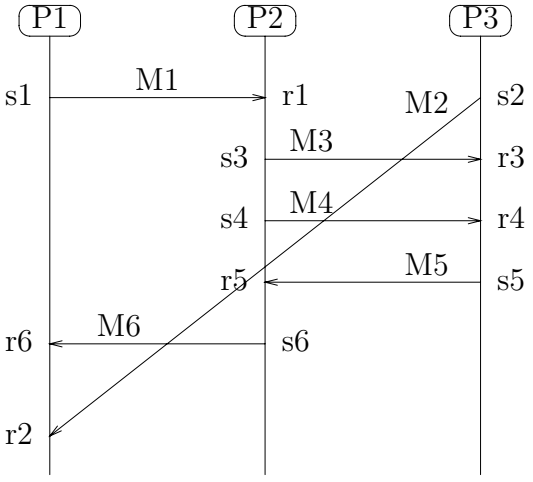


Fig. 1. Visual representation of an MSC.

Definition 1. *An MSC M over process set $\mathcal{P}$ is a tuple $\langle E, <, \mathcal{P}, \ell, t, m\rangle$:*

- *E is a set of events,*
- *$\ell : E \to [n]$ is a mapping that associates each event with a process (*location*),*
- *$t : E \to \{\mathrm{s}, \mathrm{r}, \mathrm{l}\}$ is a mapping that describes each event as send, receive or local (*type*),*
- *$m : t^{-1}(\mathrm{s}) \longrightarrow t^{-1}(\mathrm{r})$ is a bijection that pairs up send and receive events (*matching *function).*
- *$< \subseteq E \times E$ is an acyclic relation defined by:*
  - *$\ell^{-1}(i)$ is totally ordered by $<$ for every instance Pi.*
  - *For all $e, f \in E$, $m(e) = f$ implies $e < f$.*

Note that we do not require that the set of events is finite. Therefore, an MSC is either finite or infinite.

A message (e, f) consists of a pair of matching send and receive events, i.e., $m(e) = f$. We assume that communication channels are fifo. Thus, we require that whenever the messages $m(e_1) = f_1$ and $m(e_2) = f_2$ are such that $\ell(e_1) = \ell(e_2)$ and $\ell(f_1) = \ell(f_2)$, we have $e_1 < e_2$ if and only if $f_1 < f_2$.

The relation $<$ is called the *visual order* of the MSC. It can be obtained from the syntactical representation of the chart. Since $<$ is required to be acyclic its reflexive-transitive closure is a partial order on E. A total order $\preceq$ on E is called a *linearization* of $<$, if $< \subseteq \preceq$. For simplifying our notations we will also write $\leq$ for the reflexive-transitive closure of $<$. The partial order between the send and receive events of Figure 1 is shown in Figure 2. In this figure we depicted the total order on each process and only the non-redundant message arcs.

Event labels. For reasoning about linearizations of partial orders of MSCs we need event labels of the following kind. We label events by their type, location and the location of the matching event. That is, for every $e, f \in E$ such that $\ell(e) = i$, $\ell(f) = j$ and $m(e) = f$ we label e by $snd(i, j)$ and f by $rec(i, j)$. For example, both send events $s3, s4$ in Figure 1 will be labeled by $snd(2, 3)$.

The set of *send* and *receive labels*, respectively, is $\mathcal{L}_S = \{snd(i, j) \mid i, j \in [n]\}$ and $\mathcal{L}_R = \{rec(i, j) \mid i, j \in [n]\}$, respectively. Let $\mathcal{L} = \mathcal{L}_S \cup \mathcal{L}_R$ be the set of all *event labels*. If there is no risk of confusion we use the notions of *event* and *event label* interchangeably.

Remark 1. The results of this paper can be directly extended in order to include message names or local events.

Note further that event labels provide all the necessary information for computing the MSC associated with a linearization, since communication is supposed to be fifo. Thus, we associate canonically the k-th event labeled $snd(i, j)$ with the k-th event labeled $rec(i, j)$ for all k.

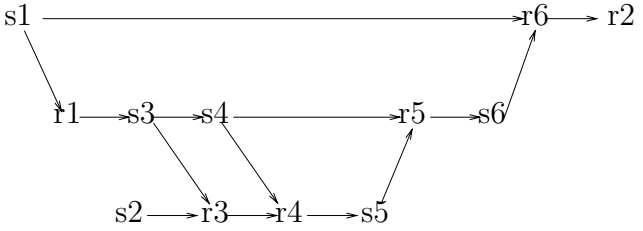


Fig. 2. The partial order between the events of the MSC in Figure 1.

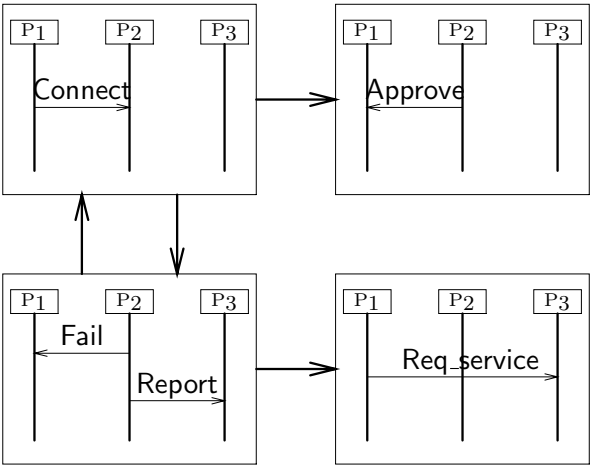
For defining more complex scenarios we need the notion of *concatenation* of two MSCs $M_1 = \langle E_1, <_1, \mathcal{P}, \ell_1, t_1, m_1\rangle$ and $M_2 = \langle E_2, <_2, \mathcal{P}, \ell_2, t_2, m_2\rangle$ over a mutual set of processes $\mathcal{P}$. The concatenation is defined if for every process Pi either $E_2 \cap \ell_2^{-1}(i) = \emptyset$ or $E_1 \cap \ell_1^{-1}(i)$ is finite. When defined, it is the MSC $M_1\, M_2 = \langle E_1 \cup E_2, <, \mathcal{P}, \ell_1 \cup \ell_2, t_1 \cup t_2, m_1 \cup m_2\rangle$ over the disjoint union of events $E_1 \cup E_2$, with the visual order given by:

$$< = <_1 \cup <_2 \cup \{(e, f) \in E_1 \times E_2 \mid \ell_1(e) = \ell_2(f)\}\,.$$

That is, the events of M_1 precede the events of M_2 for each process, respectively. Note that there is no synchronization of the different processes when moving from one MSC to the next one (*weak sequencing*). Hence, it is possible that one process is still involved in some actions of M_1, while another process has advanced to an event of M_2. We can use the concatenation repeatedly, and define infinite products $M_1 M_2 \ldots$ similarly.

The high-level description of collections of MSC scenarios provided by MSC'96 consists of a directed graph called HMSC (*high-level MSC*), where each node contains a finite MSC (see example below).

Definition 2. *An HMSC N is a tuple $\langle S, \to, s_0, c, \mathcal{P}\rangle$ where $(S, \to, s_0)$ is a finite transition system with initial state $s_0 \in S$ and states labeled by finite MSCs over $\mathcal{P}$. State $s \in S$ is labeled by the finite MSC $c(s)$. An* execution *of N is a (finite or infinite) MSC $c(s_0)\, c(s_1)\, c(s_2) \ldots$ associated with a maximal path $s_0 \to s_1 \to \cdots$ in $(S, \to, s_0)$.*



Given an HMSC N (or a set N of MSCs) we denote by $\mathrm{Lin}(N) \subseteq \mathcal{L}^\infty = \mathcal{L}^* \cup \mathcal{L}^\omega$ the set of $\mathcal{L}$-labeled linearizations of (finite or infinite) executions in N.

3 MSC Languages

In this section we consider the question whether a regular language is an MSC language, i.e., whether it is equivalent to $\mathrm{Lin}(N)$ for some (finite or infinite) set N of (finite or infinite) MSCs. Being an MSC language is a necessary condition for the HMSC equivalence problem. It turns out that testing whether a regular language is an MSC language is PSPACE-complete. Fortunately, there will be a simple syntactic condition called diamond property (Section 3.1) which ensures that an automaton accepts an MSC language.

By regular language we mean languages given by (non-deterministic) Büchi automata $\mathcal{A}$ over the set of event labels $\mathcal{L}$. The accepted language $L(\mathcal{A})$ contains all finite words labelling a path to a final state and all infinite words labelling a path that repeats infinitely often a final state.

We need some further notations. For $a \in \mathcal{L}$ and $v \in \mathcal{L}^\infty$ let $\#(v,a) \in \mathbb{N} \cup \{\infty\}$ denote the number of occurrences of symbol a in v. We write $(a,b) \in \mathcal{M}$ whenever $a, b \in \mathcal{L}$ are such that with $a = snd(i,j)$ and $b = rec(i,j)$ for some i, j. We call $w \in \mathcal{L}^\infty$ *well-formed*, if all receives in w have matching sends. Formally, for every prefix v of w, we have $\#(v,a) \geq \#(v,b)$ for all $(a,b) \in \mathcal{M}$. A finite or infinite word $w \in \mathcal{L}^\infty$ is called *complete* if it is well-formed and $\#(w,a) = \#(w,b)$ for all $(a,b) \in \mathcal{M}$.

Definition 3. *A language $L \subseteq \mathcal{L}^\infty$ over $\mathcal{L}$ is an MSC language if there is a set N of finite or infinite MSCs such that $L = Lin(N)$.*

Let $\ell(snd(i,j)) = i$ and $\ell(rec(i,j)) = j$. For $w \in \mathcal{L}^\infty$ and $i \in [n]$ we denote by $\pi_i(w)$ the projection of w on the subalphabet $\mathcal{L}_i = \{a \in \mathcal{L} \mid \ell(a) = i\}$ of event located on process Pi. The following characterization of MSCs is well-known and easy to check:

Proposition 1. *A language $L \subseteq \mathcal{L}^\infty$ over $\mathcal{L}$ is an* MSC language *if and only if it consists only of complete words and it satisfies the following property:*
For every $u, v \in \mathcal{L}^\infty$ with $\pi_i(u) = \pi_i(v)$ for all $i \in [n]$, we have $u \in L$ if and only if $v \in L$.

Suppose that a Büchi automaton $\mathcal{A}$ accepts only complete words. Then we can check the second condition of Proposition 1 in PSPACE. Specifically we test that the shuffle of $(\pi_i(L(\mathcal{A})))_{i \in [n]}$, restricted to certain complete words, is included in $L(\mathcal{A})$. It suffices to consider only complete words w such that $0 \leq \#(v,a) - \#(v,b) \leq |\mathcal{A}|$ for every prefix v of w and $(a,b) \in \mathcal{M}$. We show below that this cannot be improved, that is, this problem is PSPACE-complete. The difficulty in checking whether a Büchi automaton accepts an MSC language stems from the partial order, and not from well-formedness, as shown in the next proposition.

Proposition 2. *Let $\mathcal{A}$ be a Büchi automaton over the alphabet $\mathcal{L}$. We can check in polynomial time whether $\mathcal{A}$ accepts only complete words.*

The algorithm in Proposition 2 is based on dynamic programming (e.g., Warshall's algorithm). For each pair $(a,b) \in \mathcal{M}$ we view the automaton as a weighted graph where a (b, resp.) is replaced by 1 (by -1, resp.), and every other label by 0. Then we check that every finite path has positive weight; moreover, that any accepting path with only finitely many weights ± 1 has a prefix with weight sum zero that includes all weights ± 1.

The proof of the PSPACE lower bound is similar to the proof given in [6,8] for testing whether a regular language is closed under partial commutations.

Proposition 3. *Let $\mathcal{A}$ be a Büchi automaton over the alphabet $\mathcal{L}$. The question whether $L(\mathcal{A})$ is an MSC language is complete for PSPACE.*

3.1 Diamond Automata

We consider throughout the paper Büchi automata such that all states are accessible from the initial state. The definition below presents a syntactic condition which guarantees that a Büchi automaton where all states are final accepts an MSC language.

Definition 4 (Diamond property). *Let $\mathcal{A}$ be a Büchi automaton over $\mathcal{L}$ with initial state q_0. We say that $\mathcal{A}$ is a* diamond automaton *if for all states p, q, r and all $a, b \in \mathcal{L}$ where $p \xrightarrow{a} q \xrightarrow{b} r$ and $\ell(a) \neq \ell(b)$ there is some state s of $\mathcal{A}$ such that $p \xrightarrow{b} s \xrightarrow{a} r$ whenever one of the following conditions is satisfied:*

1. *$(a, b) \notin \mathcal{M}$, or*
2. *$(a, b) \in \mathcal{M}$ and there exists a word $v \in \mathcal{L}^*$ with $q_0 \xrightarrow{v} p$ and $\#(v, a) > \#(v, b)$.*

Note that in a diamond automaton $\mathcal{A}$ two linearizations of the same finite MSC cannot be distinguished in the sense that they reach the same set of states in $\mathcal{A}$. Of course, this kind of property is basic in concurrent systems.

Proposition 4. *Let $\mathcal{A}$ be a diamond Büchi automaton where all states are final. The restriction of $L(\mathcal{A})$ to the set of complete words is an MSC language.*

Remark 2. It is not difficult to check that the assumption on final states in the proposition above is needed for the result.

4 HMSC Languages

Some simple finite state communication protocols cannot be represented by an HMSC. The example in Figure 3 appears e.g. in [3] and describes a scenario of the alternating bit protocol. Note that whenever we match the first unmatched send event, say e_1, we are forced to add g_1, which is not yet matched. Matching g_1 adds e_2 and so on. That is, the execution in Figure 3 has no finite complete prefix. In this section we propose algorithms for checking whether a specification given as a Büchi automaton can be turned into an equivalent HMSC specification.

Definition 5 (HMSC language). *A language $L \subseteq \mathcal{L}^\infty$ is called an HMSC language if some HMSC N exists such that $L = Lin(N)$.*

We want to test whether a Büchi automaton $\mathcal{A}$ accepts an HMSC language and if this is the case, compute an equivalent HMSC. As shown in [5] we just have to require that $L(\mathcal{A})$ is a finitely generated MSC language, see Theorem 1 below. Before stating the theorem we need some further notations. For a language $L \subseteq \mathcal{L}^\infty$ of complete words we define $[L]$ as the least MSC language $K \subseteq \mathcal{L}^\infty$ with $L \subseteq K$. We call $[L]$ the *MSC closure of* L. Let $w \in \mathcal{L}^\infty$. An *MSC decomposition* of w is a sequence of complete finite words $w_1, w_2, \ldots$ such that $w \in [w_1 w_2 \cdots]$.

The theorem below was shown in [5] (Thm. 4.5) for languages of finite MSCs given by deterministic finite automata. Let $G \subseteq \mathcal{L}^*$, then we denote below by G^∞ the language $G^* \cup G^\omega$, with $G^\omega = \{w_0 w_1 \cdots \mid w_i \in G\}$.

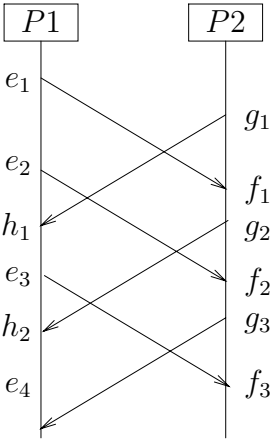


Fig. 3. A prefix of an infinite communication scenario that cannot be decomposed into finite MSCs.

Theorem 1 ([5]). *Let $\mathcal{A}$ be a Büchi automaton accepting an MSC language. Then $L(\mathcal{A})$ is an HMSC language if and only if it is* finitely generated*, i.e., if there is a finite MSC language $G \subseteq \mathcal{L}^*$ such that $L(\mathcal{A}) \subseteq [G^\infty]$. Moreover, if such G exists then there is an HMSC N of size $O(|G| \cdot |\mathcal{A}|)$ with $L(\mathcal{A}) = Lin(N)$.*

4.1 Diamond Automata and Finite Generation

The main result of [5] states that it is decidable whether a regular MSC language $L \in \mathcal{L}^*$ given by its minimal DFA is an HMSC language. Two structural properties are satisfied by minimal DFA of MSC languages, the diamond property and fixed buffer capacities. The last one means that we can associate with every state p and every pair $(a, b) \in \mathcal{M}$ a value $d(a, b)$ such that $\#(u, a) - \#(u, b) = d(a, b)$ for all words u labelling a path from the initial state to p. Furthermore, $d(a, b) < s$ where s is the size of $\mathcal{A}$. We denote a Büchi automaton satisfying this property an *automaton with fixed capacities*. Note that the state information of a finite state communication protocol usually determines the buffer sizes.

We extend and refine [5] by showing that for any diamond Büchi automaton $\mathcal{A}$ with fixed capacities we can test in polynomial time whether the accepted language is finitely generated. The next propositions describe the two cases where $L(\mathcal{A})$ is *not* finitely generated. Roughly speaking, the first case (Proposition 5) means that for a matching pair $(a, b) \in \mathcal{M}$ of events we have an arbitrary large number of events c between a and b. The second case (Proposition 6) corresponds to the situation in Figure 3.

As for MSCs there is a natural pairing between events (positions) in words. Let e be the k-th occurrence of $snd(i, j)$ and f the k-th occurrence of $rec(i, j)$ in v, then we say that e *matches* f. Let $uavbw$ be a complete word, $a \in \mathcal{L}_S$. If the position matching a belongs to the suffix bw, then we say that a *matches*

after b. Note that with fixed capacities, the position matched by a in $uavbw$ is the same as in $u'avbw$, for all u, u' labelling paths that end in the same state. For any complete word $w \in \mathcal{L}^\infty$ we define a partial order $\leq$ between events of w in the same way we defined the visual order for MSCs. It is the partial order generated by $a < b$ in $w = taubv$ where either a matches b or $\ell(a) = \ell(b)$.

Proposition 5. *Let $\mathcal{A}$ be a diamond Büchi automaton with fixed capacities. Let s denote the number of states of $\mathcal{A}$. Assume that there is some $w \in L(\mathcal{A})$ of the form*

$$w = x\, a\, y\, a_1 b_1\, w_1 \cdots w_{m-1}\, a_m b_m\, w_m\, bz$$

where $a, a_i \in \mathcal{L}_S$, $b, b_i \in \mathcal{L}_R$, $x, y, w_i \in \mathcal{L}^*$, $z \in \mathcal{L}^\infty$, and $m \leq n$ satisfy the following conditions:

1. *a matches after b, and $(a, b) \in \mathcal{M}$.*
2. *a_i matches b_i for all $1 \leq i \leq m$.*
3. *$\ell(a) = \ell(a_1)$, $\ell(b_1) = \ell(a_2), \ldots, \ell(b_{m-1}) = \ell(a_m)$ and $\ell(b_m) = \ell(b)$.*
4. *For some $1 \leq k \leq m$ we have $\sum_{e:\, \ell(e)=\ell(b_k)} \#(w_k, e) > s^2$.*

Then $L(\mathcal{A})$ is not finitely generated. Moreover, the conditions above can be checked in logarithmic space.

Proof. (Sketch.) By conditions 2,3 we have that $a < a_1 < b_1 < \cdots < a_m < b_m < b$ in w. Since a matches after b, it follows that the number of occurrences of b in $ya_1b_1w_1 \cdots a_mb_mw_m$ is at most s. By the last condition we can obtain some word w' from w by pumping a factor of w_k that contains no occurrence of b. Thus, a still matches after b in w'. For all new occurrences of e we must have $b_k < e < a_{k+1}$ in w'. Moreover, since $\mathcal{A}$ has fixed capacities, a_l still matches b_l in w' for all k. Since the earliest event matched by a is b and $a < b$, we have that a, b belong to the same factor in every MSC decomposition of w'. This means that arbitrary many events must belong to the same finite factor, contradiction.

For the complexity it suffices to guess a path and store $|x|_a - |x|_b \leq s$, checking against the number of b following a in $aya_1b_1w_1 \cdots a_mb_mw_m$. □

Proposition 6. *Let $\mathcal{A}$ be a diamond Büchi automaton with fixed capacities. Let s denote the number of states of $\mathcal{A}$. Assume that there is some $w \in L(\mathcal{A})$ of the form*

$$w = y_0 a_1\, y_1\, a_2\, x_1\, b_1\, y_2\, a_3\, x_2\, b_2 \cdots a_{t+1}\, x_t\, b_t\, z$$

where $a_i \in \mathcal{L}_S$, $b_i \in \mathcal{L}_R$, $x_i, y_i \in \mathcal{L}^*$, $z \in \mathcal{L}^\infty$ satisfy the following conditions:

1. *For all $1 \leq i \leq t$, a_i matches after b_i. Moreover, $\#(x_i, b_i) = 0$ and $(a_i, b_i) \in \mathcal{M}$ for all i.*
2. *For all $1 \leq i < t$, we have $a_i < a_{i+1} < b_i$ in w.*
3. *$t > sn$.*

Then $L(\mathcal{A})$ is not finitely generated. Moreover, the conditions above can be checked in logarithmic space.

Proof. (Sketch.) The first two conditions ensure that $a_i < a_{i+1} < b_i$ belong to the same factor in any MSC decomposition of w into MSCs. Thus, the events $a_1, \ldots, a_{t+1}, b_1, \ldots, b_t$ belong to the same MSC.

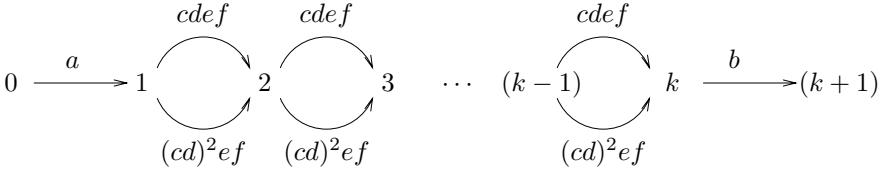
Since $t > sn$ we can assume that we have an accepting path in $\mathcal{A}$ which is labeled by $w = uxv$ and such that $x = a_i x_{i-1} b_{i-1} y_i \cdots a_{j-1} x_{j-2} b_{j-2} y_{j-1}$ is a loop where $a_{i-1} = a_{j-1}$, for some $i < j$. Consider some word $w' = uxx \cdots xv$ obtained from w by pumping x. We denote the k-th copy of x in w' as $x^{(k)}$. Assume that a_i, a_{j-1} in $x^{(k)}$ must belong to the same factor in any MSC decomposition. Note that the earliest event matched by a_{j-1} in $x^{(k)}$ is the event b_{i-1} in $x^{(k+1)}$. Suppose that these two events belong to different factors in some MSC decomposition of w'. Then the factor containing b_{i-1} must precede the factor containing a_{j-1}. Since $a_i < b_{i-1}$ in $x^{(k+1)}$, the occurrences of a_i in $x^{(k)}$, $x^{(k+1)}$ must belong to a factor preceding a_{j-1}. This contradicts the fact that a_i and a_{j-1} in $x^{(k)}$ must belong to the same factor. By this argument we obtain that $a_1, \ldots, a_{i-1}$ and the occurrences of $a_i, \ldots, a_{j-1}$ in all $x^{(k)}$ belong to the same factor in any MSC decomposition of w'. Thus, arbitrary many events must belong to the same finite MSC factor, which yields a contradiction. For the complexity it suffices to guess a path and store the difference $\#(x_{i-1}y_i, a_i) - \#(x_{i-1}y_i, b_i) \leq s$ between a_i and a_{i+1}, for all i. □

From Propositions 5 and 6 we obtain the upper bound of the following theorem:

Theorem 2. *The following question is complete for NLOGSPACE, hence solvable in polynomial time:*

Let $\mathcal{A}$ be a diamond Büchi automaton with fixed capacities. We ask whether $L(\mathcal{A})$ is finitely generated.

Remark 3. From Propositions 5 and 6 it follows that if $L(\mathcal{A}) \subseteq [G^\infty]$ for some finite MSC language $G \subseteq \mathcal{L}^*$, then G can be chosen such that all words in G have polynomial length, hence the size of G is at most exponential in s, n. This is also the upper bound for the size of an HMSC N with $\mathrm{Lin}(N) = L(\mathcal{A})$. The following example matches the exponential upper bound. Let $n = 3$ and let $a, \ldots, f$ denote the event labels $a = snd(1,2)$, $b = rec(1,2)$, $c = snd(1,3)$, $d = rec(1,3)$, $e = snd(3,2)$, $f = rec(3,2)$. The automaton below has 0 as initial state and $k+1$ as final state. It accepts a finite language of 2^{k-1} finite words. Every such word corresponds to a distinct MSC node in an equivalent HMSC.



We describe now the algorithm for computing a finite set $G \subseteq \mathcal{L}^*$ of generators for $L(\mathcal{A})$, if $L(\mathcal{A})$ is finitely generated. Let ℓ be the upper bound on the length of elements of G. For each pair of states p, q of $\mathcal{A}$ of zero capacity we consider all paths of length at most ℓ from p to q. This can be done in an BFS

way using partial-order reduction [1] (this avoids generating several linearizations of the same MSC factor). For each such path we check whether its label v is non-trivially decomposable into MSCs. If this is not the case, we add v to G. For checking whether v is decomposable we can e.g. directly adapt the following idea of [4]. With v we associate a directed graph G_v, where nodes correspond to events in v and edges (e, f) corresponding to $e < f$. Whenever e matches f we add the back edge (f, e). Then v is indecomposable if and only if G_v is strongly connected. We can compute G_v either using Tarjan's linear-time algorithm or on-line by an union-find algorithm keeping track of strongly connected components.

4.2 Bounded Automata, Communicating Automata, and Finite Generation

The diamond property required for Büchi automata in Theorem 2 plays a crucial role for the efficient test of finite generation. We show in this section that without this structural property, the question of finite generation becomes more difficult. We consider two settings, bounded automata and communicating finite-state machines.

For bounded automata we suppose that we are given a Büchi automaton $\mathcal{A}$ accepting complete words. However, we drop the assumption that $L(\mathcal{A})$ is an MSC language. Instead, we require that $\mathcal{A}$ is bounded [2,7]. This condition ensures that the MSC closure of $L(\mathcal{A})$ is still regular, see Theorem 3 below. An automaton $\mathcal{A}$ is called *bounded*, if every word v labelling a loop in $\mathcal{A}$ satisfies the following (syntactic) condition. We associate a directed graph C_v with v. The nodes of C_v are process indexes $i \in [n]$ and we have an edge from i to j if v contains the symbols $snd(i, j)$ and $rec(i, j)$. Then we require that C_v is strongly connected.

Theorem 3 ([2,7]). *Let $\mathcal{A}$ be bounded and accepting only complete words. Then the MSC language $[L(\mathcal{A})]$ is regular and there is an automaton of size in $2^{poly(s,n)}$ accepting $[L(\mathcal{A})]$.*

The question we ask is whether there is any HMSC N such that $[L(\mathcal{A})] = \mathrm{Lin}(N)$. By Theorem 1 this problem reduces to checking whether $[L(\mathcal{A})]$ is finitely generated. Equivalently, we ask whether $L(\mathcal{A})$ is finitely generated, i.e., if there exists some finite MSC language $G \subseteq \mathcal{L}^*$ such that $L(\mathcal{A}) \subseteq [G^\infty]$. If such G exists then we obtain an HMSC N for $[L(\mathcal{A})]$ of size $O(|G| \cdot |\mathcal{B}|)$, with $L(\mathcal{B}) = [L(\mathcal{A})]$.

Theorem 4. *The following question is complete for co-NP:*

Let $\mathcal{A}$ be a bounded Büchi automaton over $\mathcal{L}$, such that every loop is labeled by a complete word. We ask whether $L(\mathcal{A})$ is finitely generated.

Proof. (Sketch.) For the lower bound we reduce from 3-SAT. Let $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, with each C_i a clause with three literals over variables $x_1, \ldots, x_p$. We use m processes $P1, \ldots, Pm$ and event labels $a_i = snd(i, i+1)$, $b_i = rec(i, i+1)$, for all $1 \leq i \leq m$ with $m+1 = 1$. The automaton $\mathcal{A}$ has $p+1$ states $q_0, \ldots, q_p$

(for simplicity we label edges by words.) There are two edges from q_i to q_{i+1}, for $0 \leq i < p$. The first edge is labeled by the word $a_{i_1} \cdots a_{i_k}$, where $C_{i_1}, \ldots, C_{i_k}$ are all clauses where the variable x_{i+1} occurs positively. The second edge is labeled by the word $a_{j_1} \cdots a_{j_l}$, where $C_{j_1}, \ldots, C_{j_l}$ are all clauses where the variable x_{i+1} occurs negated. That is, any path from q_0 to q_p corresponds to a variable assignment. Moreover, the send labels occurring on the path to q_p correspond to clauses which are satisfied by the assignment. Finally, there is one self-loop around q_p, labeled by $a_1^3 b_1^3 \cdots a_m^3 b_m^3$. Clearly, $\mathcal{A}$ is bounded.

Consider a maximal path of $\mathcal{A}$ starting in q_0 and the associated MSC M. It is easy to see that if the first event of each process Pi, $i > 1$, is a send a_i and $P1$ starts with more than three occurrences of a_1, then M is not decomposable as product of finite MSCs. This occurs when F is satisfied by the assignment. For the converse we show that the blocks of 3 consecutive b_i suffice for a decomposition in finitely many finite MSC factors.

For the upper bound we show that any automaton that accepts a non-finitely generated language satisfies variants of Propositions 5 and 6, where the length of the counter-example is polynomial in the size of the automaton s and the number of instances n. For this bound we use that buffers are bounded by the size of the automaton. □

Not surprisingly, we can show that allowing unbounded message queues makes the problem of HMSC equivalence undecidable. The automaton model used in the proposition below are communicating finite state machines with fifo buffers.

Proposition 7. *It is undecidable whether a network of communicating finite state processes is equivalent to an HMSC.*

References

1. R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, and S. K. Rajamani. Partial order reduction in symbolic state space exploration. In *Proc. of CAV'97*, LNCS 1254, pp. 340–351, 1997.
2. R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. of CONCUR'99*, LNCS 1664, 1999.
3. E. Gunter, A. Muscholl, and D. Peled. Compositional message sequence charts. In *Proc. of TACAS'2001*, LNCS 2031, pp. 496–511, 2001.
4. L. Hélouët and P. Le Maigat. Decomposition of Message Sequence Charts. In *Proc. of the 2nd Workshop on SDL and MSC (SAM'2000)*, pages 46–60, 2000.
5. J. G. Henriksen, M. Mukund, K. Narayan Kumar, and P. Thiagarajan. On message sequence graphs and finitely generated regular MSC languages. In *Proc. of ICALP'2000*, LNCS 1853, pages 675–686, 2000.
6. A. Muscholl. *Über die Erkennbarkeit unendlicher Spuren.* Teubner, 1996.
7. A. Muscholl and D. Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In *Proc. of MFCS'99*, LNCS 1672, pp. 81–91, 1999.
8. D. A. Peled, T. Wilke, and P. Wolper. An algorithmic approach for checking closure properties of temporal logic specifications and omega-regular languages. *Theoretical Computer Science*, 195(2):183–203, 1998.

Fractional Path Coloring with Applications to WDM Networks*

Ioannis Caragiannis[1], Afonso Ferreira[2], Christos Kaklamanis[1], Stéphane Pérennes[2], and Hervé Rivano[2]

[1] Computer Technology Institute and
Department of Computer Engineering and Informatics
University of Patras, 26500 Rio, Greece
{caragian,kakl}@cti.gr

[2] MASCOTTE Project
I3S & INRIA Sophia Antipolis
B.P. 93, 06902 Sophia Antipolis Cedex, France
{Afonso.Ferreira,Stephane.Perennes,Herve.Rivano}@sophia.inria.fr

Abstract. This paper addresses the natural relaxation of the path coloring problem, in which one needs to color directed paths on a symmetric directed graph with a minimum number of colors, in such a way that paths using the same arc of the graph have different colors. This classic combinatorial problem finds applications in the minimization of the number of wavelengths in wavelength division multiplexing (WDM) all–optical networks.

1 Introduction

Graph coloring (or vertex coloring) is a fundamental problem of Computer Science. Given a graph, the graph coloring problem is to assign colors to vertices in such a way that adjacent vertices are assigned different colors and the number of colors used is minimized.

In general, graph coloring is very hard to solve to optimality or even to approximate. A related problem, the *path coloring* problem, consists in coloring a set $\mathcal{P}$ of paths on a graph $\mathcal{G}$ so that two paths sharing an edge of $\mathcal{G}$ have different colors. This problem is equivalent to coloring the corresponding *conflict graph*, i.e., the graph whose vertices represent the paths of $\mathcal{P}$ and where there is an edge between vertices representing conflicting paths. Notice, however, that this problem was proven to be the same as standard graph coloring in terms of complexity or difficulty to approximate, since any n-vertex graph is the conflict graph of a set of paths of an $n \times n$ grid [CGK92]. Furthermore, Tarjan proved it to be $\mathcal{NP}$-hard even for trees [Tar85].

* This work was supported in part by the European Union under IST FET Project ALCOM–FT and Improving RTN Project ARACNE. Afonso Ferreira and Stéphane Pérennes are researchers with the French CNRS.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 732–743, 2001.

Our work is motivated by applications in wavelength division multiplexing (WDM) optical networks and call scheduling, which has recently triggered a renewed interest in path coloring on special classes of graphs. In such applications, one is given an optical network with n nodes and a multiset of point-to-point communication requests, and must assign to each request a lightpath and to each lightpath a color (wavelength) so that *conflicting* lightpaths (i.e., lightpaths using the same link) are assigned different colors. The goal is to minimize the number of colors used. This problem (known as the wavelength routing problem) has been widely studied in the literature [Tuc75,BGP+96,EJK+99,ACKP00] . It has been proved to be difficult ($\mathcal{NP}$-hard even for rings); moreover one can show that there exist networks with $O(n^2)$ vertices and n requests on which it is hard to decide if the optimal number of colors is either 1 or n; thus, in general, the problem is also hard to approximate.

In this paper we study the case where the lightpaths have already been assigned to requests. It is then clear that the above problem is reduced to path coloring, with the only difference that the underlying graph is usually *directed* (since optical transmissions are one-way) and symmetric. Note that there are important differences between the directed and the undirected version of the problem; for instance, the problem can be solved in polynomial time in symmetric directed stars, but is in $\mathcal{NP}$-hard for undirected stars [Bea00].

In the rest of the paper we shall focus on the directed version of the path coloring problem. Unless otherwise specified, we shall use the terms paths and graphs to denote directed paths and symmetric directed graphs, respectively.

Previous work. Recently, several papers studied path coloring on simple networks like meshes, rings and trees [BGP+96,GHP97,Kum98,EJK+99]; in these topologies, the problem has been proved to be $\mathcal{NP}$-hard [EJ01]. Most of the results study the relationship between the load of the set of paths (i.e., the maximum number of paths crossing an arc) and the number of colors of an optimal coloring. Notice that the load is a lower bound for the optimal number of colors.

For rings (where the problem is indeed the classical circular arc coloring problem), a $\frac{3}{2}$ approximation was proposed in [Kar80]; this approximation ratio was recently improved to a $1 + \frac{1}{e} \approx 1.37$ by Kumar [Kum98]. This latter result exploits a reduction of the circular arc coloring problem to a special instance of integral multicommodity flow problem due to Tucker [Tuc75]. Indeed, the coloring of [Kum98] is obtained by, first, solving the multicommodity flow problem with fractional numbers and then, by performing a randomized rounding of this solution.

For trees, Erlebach *et al.* [EJK+99] present a polynomial time deterministic algorithm which colors any set of paths of load π using at most $5\pi/3$ colors. This algorithm is greedy in the following sense: it proceeds in phases, one per each node v of the tree. The nodes are considered following their breadth–first numbering. The phase associated with node v, assumes that there is already a partial proper coloring where all paths that touch (i.e., start, end, or go through) nodes with numbers strictly smaller than v's have been colored and no other path

has been colored. It has also been proved that this algorithm is optimal within the class of greedy deterministic algorithms [EJK+99].

Recently, Auletta *et al.* [ACKP00] used a different approach for binary trees. Instead of computing greedily a solution from top to bottom, the algorithm actually keeps a distribution of solutions (it computes from top to bottom one random element of the distribution). With high probability, an element of the distribution has the particularity to look locally random (i.e., in each 3-star). This implies some kind of "average case" of the greedy deterministic algorithm and yields an improvement of the approximation ratio. The algorithm colors any set of paths of load π on a binary tree using at most $7\pi/5 + o(\pi)$ colors, with high probability. The hidden constants in the low order term are huge and are due to the integrality constraints of the problem, and the random choices of the algorithm.

Note that these two results approximate the optimal number of colors within $5/3$ and $7/5 + o(1)$, respectively.

Our results. Our approach is based on the fact that the graph coloring problem is equivalent to assigning unit cost to some of the independent sets of the graph such that all vertices are covered (i.e., are contained in an independent set of unit cost) and the total cost (i.e., the number of independent sets of unit cost) is minimized.

We then observe that Kumar's fractional solution [Kum98] gives in fact an optimal *fractional coloring* [GLS81], a natural relaxation of graph coloring, where the independent sets covering the vertices of the graph may have fractional weights. Notice, however, that this relaxation is generally also hard to approximate.

The positive side of our observation is that it allows us to prove several results related to integral and fractional path coloring, as follows.

- In Section 3 we show that fractional path coloring in bounded–degree trees can be solved in polynomial time. This result is constructive, i.e., our algorithm inductively builds a polynomial size linear program whose final solution is such an optimal fractional coloring.
- This fractional path coloring algorithm for trees can be easily adapted to any bounded–degree and bounded–treewidth graph, as described in Section 4.1. Also, extending our techniques, we characterize polynomially solvable instances of the fractional path coloring problem in general graphs in Section 4.2.
- We show an upper bound of $7\pi/5$ on the fractional path chromatic number in binary trees in terms of the load π of the set of paths (see Section 4.3). This is somewhat related to the results in [ACKP00] since, their random algorithm can be seen as an attempt to emulate a balanced fractional coloring and, on the other hand, a balanced fractional coloring can be interpreted as a perfect random sample. However, our fractional analysis is much simpler, our algorithm is deterministic, and our bound is tighter.
- With respect to integral path coloring, in Section 4.4 we provide a randomized approximation algorithm for path coloring in bounded–degree trees

with approximation ratio $1.61 + o(1)$. This is done by applying randomized rounding to the fractional solution presented in Section 3.

In the next section we recall the formal definitions of (fractional) coloring and (fractional) path coloring.

2 Fractional Coloring

The graph coloring problem can be considered as finding a minimum cost integral covering of the vertices of a graph by independent sets of unit cost. Given a graph $\mathcal{G} = (V, E)$, this means solving the following integer linear program:

$$\begin{array}{rll} \text{minimize} & \sum_{I \in \mathcal{I}} x(I) & \\ \text{subject to} & \sum_{I \in \mathcal{I}: v \in I} x(I) \geq 1 & \forall\ v \in V \\ & x(I) \in \{0, 1\} & \forall\ I \in \mathcal{I} \end{array}$$

where $\mathcal{I}$ denotes the set of the independent sets of $\mathcal{G}$.

This formulation has a natural relaxation into the following linear program:

$$\begin{array}{rll} \text{minimize} & \sum_{I \in \mathcal{I}} \bar{x}(I) & \\ \text{subject to} & \sum_{I \in \mathcal{I}: v \in I} \bar{x}(I) \geq 1 & \forall\ v \in V \\ & 0 \leq \bar{x}(I) \leq 1 & \forall\ I \in \mathcal{I} \end{array}$$

The corresponding combinatorial problem is called the *fractional coloring problem* (see [GLS81]), and the value of an optimal solution is called the *fractional chromatic number* $w_f(\mathcal{G})$.

If $\bar{x}$ is a valid cost function over the independent sets of the graph $\mathcal{G}$, we call it a *fractional coloring* of $\mathcal{G}$. We use the symbol $\bar{x}(\mathcal{G})$ to denote the cost of the solution $\bar{x}$.

In general, the fractional chromatic number is as hard to approximate as the chromatic number since a classical result states that any ρ–approximation of the fractional chromatic number leads to a $\rho \log(n)$–approximation of the chromatic number. Indeed, the size of the above described linear problem is exponential (proportional to the number of independent sets of $\mathcal{G}$).

It is well–known [GLS81] that the dual of the above linear program is the following:

$$\begin{array}{rll} \text{maximize} & \sum_{v \in V} y(v) & \\ \text{subject to} & \sum_{v \in I} y(v) \leq 1 & I \in \mathcal{I} \\ & y(v) \geq 0 & \end{array}$$

In this problem, a non–trivial constraint is violated if and only if the weight of one independent set I, defined as the sum of the weights of its vertices $\sum_{v \in I} y(v)$, is greater than 1. Hence, the maximum weighted independent set problem (MWIS) is a separation oracle for this last problem. According to the separation and optimization equivalence (see [GLS93], Th. 4.2.7, page 106, and [GLS81]), the dual of the fractional coloring and the MWIS are equivalent up to polynomial

reduction. Thus, computing the fractional chromatic number is polynomially equivalent to solving MWIS, which is polynomial for trees [Gar94]. However, this duality argument does not provide any effective fractional coloring algorithm but rather a way to compute the fractional chromatic number.

We now extend some terms of graph coloring to path coloring. Given a set of paths $\mathcal{P}$ on a graph $\mathcal{G}$, we define an *independent set of paths* as a set of pairwise arc–disjoint paths, i.e., a set of paths whose corresponding vertices form an independent set of the conflict graph. If $\mathcal{G}_c$ is the conflict graph of $\mathcal{P}$ on $\mathcal{G}$, we will denote by $w(\mathcal{G}, \mathcal{P})$ ($w_f(\mathcal{G}, \mathcal{P})$) the (fractional) chromatic number of $\mathcal{G}_c$ and call it the (fractional) path chromatic number of $\mathcal{P}$ on $\mathcal{G}$. We will also denote by $\pi(\mathcal{G}, \mathcal{P})$ the load of $\mathcal{P}$ on $\mathcal{G}$. i.e. the maximum number of paths of $\mathcal{P}$ using any fixed edge of $\mathcal{G}$.

As far as the relationship between w and w_f is concerned, the following result is implicit in the work of Kumar [Kum98] which addresses the case of the ring:

$$w(C_n, \mathcal{P}) \leq w_f(C_n, \mathcal{P}) + \frac{\pi(C_n, \mathcal{P})}{e} + o(\pi(C_n, \mathcal{P})).$$

In the very specific case where the conflict graph is a proper circular arc graph, Niessen and Kind proved that $w = \lceil w_f \rceil$ in [NK98].

In the next section we show that optimal fractional path colorings in bounded–degree trees can be computed by solving a polynomial–size linear program. Our technique can be easily extended to networks of bounded degree and bounded treewidth. As an application (Section 4), we adapt the idea of producing path colorings by applying the randomized rounding technique to fractional path colorings and obtain 1.61– and 2.22–approximation algorithms for path coloring in bounded–degree trees, and trees of rings, respectively.

3 An Algorithm for Fractional Path Coloring in Trees

In this section we study the fractional path coloring problem in trees. We assume that paths are labeled so that labels are unique, and given a set of paths we will denote $\mathcal{P}(p)$ the path of family $\mathcal{P}$ which has label p. The special label $\emptyset$ will be associated to a non-existent or void path.

3.1 Trace of a Fractional Coloring

Given a solution $\bar{x}$ for path coloring that is a weight function $\bar{x}$ on the independent sets of the tree T, we define the *trace of the fractional coloring* $\bar{x}$ *on an edge* e as the following function X^e:

- For any label p, let $\mathcal{I}(p)$ be the set of independent sets containing $\mathcal{P}(p)$, and $\mathcal{I}(p, q) = \mathcal{I}(p) \cap \mathcal{I}(q)$.
- For any pair of label p, q such that $\mathcal{P}(p)$ and $\mathcal{P}(q)$ use e in opposite directions, let $X^e(p, q) = \sum_{I \in \mathcal{I}(p,q)} \bar{x}(I)$.

- For any label p such that $\mathcal{P}(p)$ uses e, let $\mathcal{I}^e(p,\emptyset)$ be the set of independent sets of $\mathcal{I}(p)$ using e in only one direction, and let $X^e(p,\emptyset) = \sum_{I\in\mathcal{I}^e(p,\emptyset),p\in I} \bar{x}(I)$.
- Finally let $\mathcal{I}^e(\emptyset,\emptyset)$ be the set of independent sets not using e.

Given an instance of fractional path coloring $(T,\mathcal{P})$, we denote by $Sol((T,\mathcal{P}),c)$ the set of all the fractional coloring of $(T,\mathcal{P})$ with cost less than c.

3.2 Split and Merge

Our algorithm inductively constructs a polynomial size linear program whose solution provides a fractional coloring. Our induction is based on *merge* and *split* operations which allow to build any instance of fractional path coloring starting from instances on stars (i.e. trees of height 1) and merging them step by step.

Consider a fractional path coloring of $(T,\mathcal{P})$ where T is not a star and such that $e = [u,v]$ is a non-terminal edge of T. The splitting of T on edge e is two smaller instances of fractional path coloring $(T_1,\mathcal{P}_1)$ and $(T_2,\mathcal{P}_2)$ defined as follows (cf. Fig. 1):

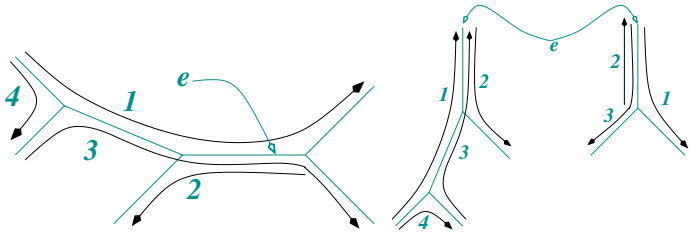


Fig. 1. Splitting a tree on e.

- Let U_1 and U_2 be the two connected components of $T \setminus \{u,v\}$, and let $T_1 = U_1 \cup \{[u,v]\}$, $T_2 = U_2 \cup \{[u,v]\}$.
- Let $\mathcal{P}_1 = \mathcal{P} \cap T_1$, $\mathcal{P}_2 = \mathcal{P} \cap T_2$ (i.e in each subtree each original path is replaced by the path it induces with a label equal to the original one).

Note that the split operation can be easily reverted by a *merge* one if one keeps track of the paths labels and of the edge where the split occurred. In what follows, we will always assume that $(T,\mathcal{P})$ can be split on the edge e into $(T_1,\mathcal{P}_1)$ and $(T_2,\mathcal{P}_2)$.

Now, we prove that one can build $Sol((T,\mathcal{P}),c)$ from $Sol((T_1,\mathcal{P}_1),c)$ and $Sol((T_2,\mathcal{P}_2),c)$.

Proposition 1 *The elements of* $Sol((T,\mathcal{P}),c)$ *are obtained by merging elements of* $Sol((T_1,\mathcal{P}_1),c)$ *and* $Sol((T_2,\mathcal{P}_2),c)$ *having the same trace on* e.

Proof: First we remark that any fractional coloring of T with cost c induces fractional colorings on $(T_1,\mathcal{P}_1)$ and $(T_2,\mathcal{P}_2)$ with cost less than c. From construction both have the same trace on e.

Conversely, given two fractional colorings $\bar{x}_1 \in Sol((T_1,\mathcal{P}_1),c)$ and $\bar{x}_2 \in Sol((T_2,\mathcal{P}_2),c)$ having the same trace X_1^e and X_2^e on e, we merge them into a fractional coloring in $Sol((T,\mathcal{P}),c)$ by repeating the following procedure until $X_1^e(p,q) = 0, \forall p,q$:

- If for some pair of labels p and q (including the $\emptyset$ label) $X_1^e(p,q) > 0$, then since $X_1^e(p,q) = X_2^e(p,q)$ there exists $I_1 \in \mathcal{I}_1(p,q)$ and $I_2 \in \mathcal{I}_2(p,q)$ with $\bar{x}_1(I_1) > 0$ and $\bar{x}_2(I_2) > 0$.
- Let $\bar{x}_{min} = min(\bar{x}_1(I_1), \bar{x}_2(I_2))$.
- Let $I = I_1 \setminus \{\mathcal{P}_1(p), \mathcal{P}_1(q)\} \cup I_2 \setminus \{\mathcal{P}_2(p), \mathcal{P}_2(q)\} \cup \{\mathcal{P}(p), \mathcal{P}(q)\}$, and note that I is an independent set of T
- Increase $\bar{x}(I)$ by $\bar{x}_{min}$ decrease $\bar{x}_1(I_1)$ and $\bar{x}_2(I_2)$ by $\bar{x}_{min}$.

To verify the claim, just note that the procedure preserves the following invariant:

- trace equality $(X_1^e = X_2^e)$
- Paths are covered either by $\bar{x}_1$, $\bar{x}_2$ or $\bar{x}$
- $\bar{x}(T,\mathcal{P}) + \bar{x}_1(T_1,\mathcal{P}_1) = \bar{x}(T,\mathcal{P}) + \bar{x}_2(T_2,\mathcal{P}_2) = c$

It follows that, at the end, $\bar{x}$ is a fractional coloring of $(T,\mathcal{P})$ with cost c. Moreover, for $i = 1,2$ the trace of $\bar{x}$ on any edge of T equals the one of $\bar{x}_i$ in T_i. □

Corollary 2 *We can compute a polynomial size linear program whose solutions are valid traces of elements of* $Sol((T,\mathcal{P}),c)$

Proof: We first assume that such a program does exist for bounded degree d stars, a naive way to get one is to use *labeled independent set variables*, that is one variable for any possible labeled independent set in a bounded degree star (as paths are labeled, we must distinguish between similar path having different labels). Hence, if the load is π, at most π^{2d} different labeled independent sets can correspond to an unlabeled independent set. Hence a pessimistic estimate count about $M(d)\pi^{2d}$ variables, where $M(d)$ is the number of perfect matchings in $K_{d,d}$. Fractional coloring is trivially described from these variables, then the trace variables are simple sum of subsets of the labeled independent set variables.

We then use an inductive algorithm to generate a linear program whose solutions are traces of $Sol((T,\mathcal{P}),c)$. Assume that T can be split on edge e into

$(T_1, \mathcal{P}_1)$ and $(T_2, \mathcal{P}_2)$ and let S_i be the linear program for traces of $Sol((T_i, \mathcal{P}_i), c)$, where we assume that the trace on e is associated to the variables $X_i^e(p, q)$. Then a system for $(T, \mathcal{P})$ is:

$$S = S_1 \cup S_2 \cup \{X_1^e(p, q) = X_2^e(p, q) \mid \forall p, q\}$$

□

Proposition 3 *Fractional path coloring in bounded–degree trees can be reduced to solving a polynomial size linear program.*

Proof: We simply have to show how to compute an element of $Sol((T, \mathcal{P}), c)$ from a trace $\mathcal{X} = \{X^e, \forall e \in T\}$ of an element of $Sol((T, \mathcal{P}), c)$ (obtained from Corollary 2). Again we proceed inductively: we start from stars and find for each one a fractional coloring having the trace that $\mathcal{X}$ induces on it. Then if $(T, \mathcal{P})$ can be split into $(T_1, \mathcal{P}_1)$ and $(T_2, \mathcal{P}_2)$, Proposition 1 provides a way to merge fractional colorings of $(T_1, \mathcal{P}_1)$ and $(T_2, \mathcal{P}_2)$ when their traces are equal, and this is the case since they are subsets of $\mathcal{X}$. □

3.3 Reducing the Problem Size

Note that our first model induces very large systems since we could get $O(n\pi^6)$ variables for binary trees, and $O(n\pi^8)$ for ternary ones. In this section we show how to reduce the system size to $O(dM(d)\pi^2 n)$, that is a size of order $O(n\pi^2)$ for bounded degree trees (with still a huge constant).

Proposition 4 *Fractional path coloring in bounded-degree d trees can be reduced to solving a linear program of size $O(dM(d)\pi^2 n)$.*

Proof: First and without loss of generality we assume that the load is uniform.

Corollary 2 shows that the size of the linear program of the fractional path coloring of $(T, \mathcal{P})$ of cost less than c is $O(n\pi^2)$ plus the sum of the sizes of the linear programs for each star of the tree T, which is less than n times the size for a star of degree d.

We use a flow-like description for the fractional path coloring problem of a star which can be related to a well-known property: the matching polytope of a bipartite graph is encoded by simple flow equations (or, equivalently, probability matrices are convex combination of permutation matrices).

Let us consider a star $\mathcal{S} = \{v_0, v_1, \ldots, v_d\} \subseteq T$ where v_0 is the center of $\mathcal{S}$ and d its degree. Arcs of $\mathcal{S}$ are $\{(v_0, v_i), (v_i, v_0), \forall i \in [1, \ldots, d]\}$. Let $\mathcal{M}$ be the set of load 1 independent sets of $\mathcal{S}$ (i.e. subsets of paths in $\mathcal{S}$ loading each arc exactly once). For all $M \in \mathcal{M}$, we build an auxiliary flow problem F_M:

- For each path $\mathcal{P}(p) \in T$ including a path of M, we add a vertex $V(p)$.
- For each pair of vertices $V(p), V(q)$ such that $\exists\, i \in [1, \ldots, d] \mid (v_0, v_i) \subseteq \mathcal{P}(p)$ and $(v_i, v_0) \subseteq \mathcal{P}(q)$, we add the arc $(V(p), V(q))$.

- We define a flow function[1] f_M on the arcs which induces a flow function f'_M on the vertices.
- For each leaf v_i of the star we add the constraint

$$\sum_{p|(v_0,v_i)\subseteq\mathcal{P}(p)} f'_M(V(p)) = c_M$$

Any solution of the previous system induces a covering of the paths with weight $f'(p)$ and a subpart of the trace equals to $f(p,q)$.

The total system for a star is obtained as follows:

- $F_M, M \in \mathcal{M}$: are considered all together.
- $X(p,q) = \sum_{M\in\mathcal{M}} f_M(p,q)$: computation of the trace variables.
- $\sum_{M\in\mathcal{M}} f'_M(p) \geq 1$: covering of the paths.
- $\sum_{M\in\mathcal{M}} c_M \leq c$: cost constraint.

Due to the above mentioned property of the matching polytope, this system describes the fractional coloring problem for a star with an encoding of the trace variables.

It follows the fractional coloring of each stars of T can be described with a system of size at most $M(d)(= |\mathcal{M}|)$ times $\pi^2 d$. Hence the system for T is of size $O(dM(d)\pi^2 n)$. □

Notice that an optimal fractional coloring can be computed in polynomial time even if the degree of the tree is $d = O(\max\{\log\pi/\log\log\pi, \log n/\log\log n\})$, since $M(d) = O(d!)$.

4 Extensions and Applications

In this section we extend the technique described in the previous section to graphs of bounded degree and bounded treewidth. We also characterize instances in arbitrary networks which can be solved in polynomial time. Furthermore, we obtain an algorithm for fractional path coloring on binary trees with cost at most $7\pi/5$. Using the results for the fractional path coloring, we achieve improved approximation algorithms for the path coloring problem in (bounded–degree) trees and trees of rings and upper bounds for the path chromatic number in terms of the fractional path chromatic number and the load.

4.1 Graphs of Bounded Degree and Bounded Treewidth

In the case of graphs of bounded degree and bounded treewidth, we obtain the following:

[1] By flow function we mean conservative for vertices. One can note that this function is a circulation since no sink and no source is present

Proposition 5 *Fractional path coloring can be solved in polynomial time in graphs of bounded degree and bounded treewidth.*

Proof sketch: The proof follows similar lines with the one for trees. Now, split and merge operations are applied to cuts of the graph. Since treewidth is bounded, we can consider cuts in which the number of edges is upper–bounded by a constant. The trace variables are defined for each edge of the cut, leading to $O(\pi^{2k})$ trace variables, where k is bounded by the treewidth. □

4.2 Some Polynomial Instances

Based on the technique described in Section 3, we can characterize instances of the path coloring problem in general graphs which can be solved in polynomial time.

Note that our approach in Section 3 can be considered as dynamic programming where one maintains a polynomial encoding of the valid traces. We express the result in terms of the number of non–isomorphic paths crossing any cut of a graph. Recall that two paths are called isomorphic if they share the same (directed) edges of the graph.

Proposition 6 *If the number of non–isomorphic paths crossing any cut of a graph is bounded, the fractional path coloring problem can be solved in polynomial time.*

Proof sketch: Assuming that the number of non–isomorphic paths crossing any cut of the graph is upper–bounded by a constant k, the trace for each cut we consider can be encoded with at most $O(\pi^k)$ trace variables. □

Note that this result depends only on properties of the set of paths; not on the underlying graph.

4.3 Fractional Path Coloring on Binary Trees

Fractional coloring of binary trees can be performed using a particular coloring, called *balanced coloring*, where the traces depend only on the number of paths going trough an arc (in some sense it can be seen as a perfect random sample). Proposition 3 states that the paths of the tree can be colored independently (using fraction of colors) in each 3-star, so that one can find a global fractional coloring consistent with the local colorings.

A simple but exhaustive analysis shows that 3-stars can be colored in a balanced way with at most $\frac{7}{5}\pi$ colors. We then obtain that, given a set of paths $\mathcal{P}$ on a binary tree T, the algorithm computes a fractional path coloring of $\mathcal{P}$ with cost at most $\frac{7\pi(T,\mathcal{P})}{5}$.

Proposition 7 *For any set of paths $\mathcal{P}$ of load $\pi(T,\mathcal{P})$ on a binary tree T, there exists a fractional coloring of cost $\frac{7\pi(T,\mathcal{P})}{5}$. Moreover, such a fractional coloring can be computed in polynomial time.*

As a corollary, we obtain that given a set of paths $\mathcal{P}$ on a binary tree T, the maximum independent set of paths of $\mathcal{P}$ has size at least $\frac{5|\mathcal{P}|}{7\pi(T,\mathcal{P})}$.

4.4 Integral Path Coloring in Trees

In this section we present an important application of our methods to the path coloring problem in trees.

Note that the result of [EJK+99] states that there exists a polynomial time algorithm which colors any set of paths $\mathcal{P}$ of load $\pi(T, \mathcal{P})$ on a tree T with at most $5\pi(T, \mathcal{P})/3$ colors. Since the load $\pi(T, \mathcal{P})$ is a lower bound for the optimal number of colors, this gives a 5/3–approximation algorithm. In the following we exploit the (optimal) solution for the fractional path coloring which can be obtained in polynomial time for bounded–degree trees to design a randomized algorithm with better approximation ratio.

Given a solution $\bar{x}$ of the fractional path coloring of the set of paths $\mathcal{P}$ on a tree T, the idea is to perform a randomized rounding to $\bar{x}$ and obtain an integral solution x. After rounding, x is not a feasible solution to the path coloring problem since some of the constraints of the form $\sum_{I \in \mathcal{I}: p \in I} x(I) \geq 1$ may be violated. However, this is a feasible solution to the path coloring problem on the set of paths $\mathcal{P}' \subseteq \mathcal{P}$, defined as the set of paths contained in independent sets I such that $x(I) = 1$. This means that we have properly color some of the paths of $\mathcal{P}$ with $w_f(\mathcal{T}, \mathcal{P})$ colors.

Following the analysis of [Kum98], we can show that if $\pi(T, \mathcal{P}) = \Omega(\log n)$, where n is the number of vertices in T, then after the rounding procedure the load of paths in $\mathcal{P} \backslash \mathcal{P}'$, i.e., the load of the paths not colored, is

$$\pi(T, \mathcal{P} \backslash \mathcal{P}') \leq \frac{\pi(T, \mathcal{P})}{e} + o(\pi(T, \mathcal{P}))$$

with high probability. Now, we can apply the algorithm of [EJK+99] to color the paths in $\mathcal{P} \backslash \mathcal{P}'$ with $\frac{5\pi(T,\mathcal{P})}{3e} + o(\pi(T, \mathcal{P}))$ additional colors. In total, we use at most

$$w_f(T, \mathcal{P}) + \frac{5\pi(T, \mathcal{P})}{3e} + o(\pi(T, \mathcal{P}))$$

colors. Since $\pi(T, \mathcal{P}) \leq w_f(T, \mathcal{P}) \leq w(T, \mathcal{P})$, we obtain the following results.

Proposition 8 *There exist a randomized* $1.61 + o(1)$*–approximation algorithm to the path coloring problem in bounded–degree trees.*

Corollary 9 *For any set of paths* $\mathcal{P}$ *on a tree* T*, it holds:*

$$w(T, \mathcal{P}) \leq w_f(T, \mathcal{P}) + \frac{5\pi(T, \mathcal{P})}{3e} + o(\pi(T, \mathcal{P})).$$

We can also apply similar arguments to bounded–degree trees of rings to obtain a 2.22–approximation algorithm.

5 Conclusions

Our research in this paper was motivated by questions related to the design of wavelength division multiplexing optical networks. We developed a new approximation tool for WDM networks by using the classical fractional coloring. Many

applications of our techniques to WDM networks can be foreseen, as in branch and bound methods, or even in the design of multifiber networks .

One intriguing open problem is to prove better bounds on the size of the gap between the cost of integral and fractional path coloring in trees. We conjecture that this gap is small.

References

[ACKP00] V. Auletta, I. Caragiannis, C. Kaklamanis, and P. Persiano. Randomized path coloring on binary trees. In *Proc. of the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'00)*, LNCS 1913, Springer, pp. 60–71, 2000.

[Bea00] B. Beauquier. *Communications dans les réseaux optiques par multiplexage en longueur d'onde*. PhD thesis, Université de Nice-Sophia Antipolis, 2000.

[BGP+96] J-C. Bermond, L. Gargano, S. Pérennes, A. A. Rescigno, and U. Vaccaro. Efficient collective communication in optical networks. In *Proc. of the 23rd International Colloquium on Automata, Languages and Programming (ICALP'96)*, LNCS 1099, Springer, pp. 574–585, 1996.

[CGK92] I. Chlamtac, A. Ganz, and G. Karmi. Lightpath communications: An approach to high bandwidth optical WAN's. *IEEE Transactions on Communications*, 40(7):1171–1182, 1992.

[EJ01] T. Erlebach and K. Jansen. The Complexity of Path Coloring and Call Scheduling. *Theoretical Computer Science*, Vol. 255 (1-2), pp. 33-50, 2001.

[EJK+99] T. Erlebach, K. Jansen, C. Kaklamanis, M. Mihail, and P. Persiano. Optimal wavelength routing on directed fiber trees. *Theoretical Computer Science*, 221(1-2):119–137, 1999.

[Gar94] N. Garg. *Multicommodity Flows and Approximation Algorithms*. PhD thesis, Indian Institute of Technology, Delhi, April 1994.

[GHP97] L. Gargano, P. Hell, and S. Pérennes. Colouring paths in directed symmetric trees with applications to WDM routing. In *Proc. of the 24th International Colloquium on Automata, Languages and Programming (ICALP'97)*, LNCS 1256, Springer, pp. 505–515, 1997.

[GLS81] M. Grötschel, L.Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[GLS93] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2. Springer, 2nd corrected edition, 1993.

[Kar80] I. A. Karapetian. On coloring of arc graphs. *Doklady Akad. Nauk Armianskoi CCP*, 70(5):306–311, 1980. (In Russian)

[Kum98] V. Kumar. Approximating circular arc colouring and bandwidth allocation in all-optical ring networks. In *Proc. of the 1st International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'98)*, 1998.

[NK98] T. Niessen and J. Kind. The round-up property of the fractional chromatic number for proper circular arc graphs. *Journal of Graph Theory*, 1998.

[Tar85] R. E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221–232, 1985.

[Tuc75] A. Tucker. Coloring a family of circular arcs. *SIAM Journal of Applied Mathematics*, 29(3):493–502, November 1975.

Performance Aspects of Distributed Caches Using TTL-Based Consistency

Edith Cohen[1], Eran Halperin[2], and Haim Kaplan[2]

[1] AT&T Labs–Research, 180 Park Avenue, Florham Park, NJ 07932,
edith@research.att.com
[2] Department of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel,
{heran,haimk}@math.tau.ac.il

Abstract. Web objects are stored and can be requested from numerous servers, including authoritative "origin" servers and caches. Objects can be modified only by their origin servers and weak consistency with cached copies is maintained by limiting their lifetime durations. Copies fetched from origin servers are received with maximum time-to-live (TTL) that equals their lifetime duration whereas copies obtained through a cache have shorter TTLs since their *age* (elapsed time since fetched from the origin) is deducted from their lifetime duration.
A request served by a cache constitutes a *hit* if the cache has a fresh copy of the object. Otherwise, the request is considered a *miss* and is propagated to another server. Performance is measured by the number of requests constituting cache misses. It is evident that the number of cache misses depends on the age of the copies the cache receives. Thus, a cache that sends requests to another cache would suffer more misses than a cache that sends requests directly to an authoritative server. More subtly, the number of misses depends on the particular configuration of higher-level caches, e.g., whether one or more higher-level caches are used. Guided by practices for Web caching, we model and compare different configurations. We also analyze the effect of *pre-term refreshes* at high-level caches and *extended lifetimes* at low-level caches and reveal patterns that may seem counter-intuitive at first. Even though TTL-based consistency is very widely used, our work seems to be the first to formally analyze it. Our analysis yields insights and guidelines for improving the performance of Web caches.

1 Introduction

Web objects are typically associated with one authority that originates and modifies them (their *authoritative* server), but can be cached and further distributed from multiple *replicating* servers (caches). Indeed, caching and replication are widely deployed for reducing load on Web servers, network load, and user-perceived latency. Replicating servers are located at different points in the network and include reverse proxies, proxy caches, and browser caches. By and large, Web servers do not maintain per-client state and thus are not aware of all locations of cached copies of objects they serve. Currently the only widely supported and deployed mechanism for consistency of cached copies is client-driven

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 744–756, 2001.

and expiration-based: the authoritative server provides an expiration time for each copy, beyond which it must be validated or discarded. Since this mechanism is the underlying motivation for our model, we further elaborate on it.

The distribution of Web content is governed by HTTP (Hyper-Text Transfer Protocol) [4]. Each object has a URL which specifies its "location" and its authoritative server. The object is requested by sending an HTTP request and the content is sent back on the respective HTTP response. The response includes a header with important information on the object, including cache directives. The directives specify if the object can be cached, and may provide explicit expiration time or information that can be used to compute one. When an object is requested from the cache then if the cache has a *fresh* (non-expired) copy, the request is processed locally. If the cached copy is *stale* (expired), it must be validated by contacting a server with a fresh copy. To this end, HTTP provides conditional `GET` requests. Similarly, if there is no cached copy, the cache must obtain a fresh copy. Requests for which the cache does not have a fresh copy and thus must contact an external server are termed *cache misses*.

An HTTP-compliant cache [4,1,7] calculates from the header fields of an object a *freshness lifetime*, which is the time interval during which the copy remains fresh (non-expired) since it left its authoritative server. The freshness lifetime is typically fixed for each object. The cache also determines from the headers the *age* of the copy, which is the elapsed time since it left its authoritative server (and resided in caches). If the age is smaller than the freshness lifetime then by subtracting the age of the copy from its freshness lifetime the cache obtains a *time-to-live (TTL)* duration for the copy (during which it is considered fresh). If the age is larger than the freshness lifetime, the object is considered stale and its TTL is zero.

Web caches are placed both as proxy caches close to clients and reverse proxies close to Web servers [7,5]. They are also sometimes configured in hierarchies [6]. Thus, often there is more than one cache placed between the end user and the origin server. Different copies corresponding to the same URL and residing in different caches can have different ages and thus different TTLs. In particular, a copy obtained by a cache from another cache expires sooner than a copy obtained from an authoritative server. Therefore, a cache that forwards requests to another cache is likely to suffer more cache misses than a cache that forwards requests to an authoritative server. Furthermore, a cache is likely to obtain more client requests than an authoritative server since it serves older copies. So even if a cache maintains fresh copies, since the copies are aged, the cache is somewhat less effective than an authoritative server [3,2].

More generally, the miss rate also varies according to the configuration of higher-level caches. We compare different basic cache configurations according to the above notions of age, TTL, and miss rates. In particular, we show that a cache which consistently uses the same "source" cache (for each object) would incur a lower miss rate than a cache that alternates between several caches, on any sequence of requests. This suggests that when a set of caches handles the workload of a set of clients, it is better to partition the workload such that the same primary cache is used for each client-object pair.

We then use extensions of the basic configuration models that incorporate two further cache behaviors [2]. The basic models assumed that only expired copies can be refreshed. HTTP caches, however, may refresh fresh items. [1] Conceivably, a cache can deploy configured periodic pre-term refreshes (*rejuvenations*) to reduce the age of cached copies and as a result, improve the miss-rate at its client caches and consequently, the number of requests it receives. Rejuvenation increases traffic between a cache and the authoritative server but can decrease traffic between the cache and its clients (note that the benefit can be large since a single cache can serve numerous clients). As pre-term refreshes decrease the average age of cached copies, it may seem that they can only increase the effectiveness of a cache. We show, however, that this is generally not the case. In particular we show that when pre-term refreshes occur in arbitrary points in time and "out of synch" with the expiration times of previous copies, performance can degrade by up to a factor of 2, which is tight under some robust conditions. We also show however, that when used "in synch" with previous expirations, pre-term refreshes can boost performance.

Although not typically the case, different caches can use different freshness lifetime durations for the same copy. [2]. Caches that use longer lifetime durations relative to their source can improve their miss-rate albeit at the cost of increased staleness. We show that for many prototypical request sequences, the tradeoff of staleness vs. miss-rate is not convex, and that specifically, integral values of the *extension factor* (the ratio of the lifetime time durations at the cache and its source) are more effective.

We also consider worst-case behavior and show that a cache which forwards request to another cache (even one that performs pre-term refreshes) can incur twice the miss-rate of a cache that uses an authoritative source. This worst-case ratio can approach e if the cache forwards requests to different caches.

In Section 2 we present the basic and extended models. In Section 3 we provide general results for the basic models, including comparison of different cache configurations. Section 4 provides general results on pre-term refreshes and freshness-lifetime extensions. Section 5 includes worst-case analysis that provides tight bounds on the competitive ratio of different configurations. We conclude in Section 6.

2 Model

We consider a set of distributed servers that originate, cache, request, and distribute copies of an object. The object can be modified over time but only by its

[1] Such *pre-term refreshes* occur when a client request contains a *no-cache* request header. When a cache received such a request it must forward it to the origin server even if it has a fresh copy. The cache uses the response to replace or refresh its older copy of the object.

[2] The majority of HTTP requests involve objects without explicit expiration and for which the freshness lifetime is determined by a heuristic. Thus, different caches can associate different freshness lifetime values with the same copy as a result of different URL filters and selecting different values for CONF_PERCENT or CONF_MAX [7].

origin server. Each copy of an object has a designated *lifetime* duration which after it elapses, the copy can not be further distributed or used. The *lifetime* value is assigned by the origin server when the copy is served. Our analysis focuses on lifetime values that are fixed throughout time. That is, the origin always provides lifetime of T. We differentiate between *origin* (authoritative) servers and *caches* (replicating servers). The *age* of a cached copy is the elapsed time since it was obtained from an origin server. The TTL of a cached copy equals the lifetime T minus its age. If the age is larger than T, the copy is *stale*. Otherwise, the copy is *fresh*. The object can be requested from any server. Origin servers always provide a copy with zero age and thus a *time-to-live* (TTL) of T. Caches process the request by providing a local fresh copy, if there is one. The request then is considered as a *cache hit*. Otherwise, another server (see Figure 1) is contacted and the response can be used to update the cached copy. The request then is a *cache miss*. The *miss rate* of cache is the fraction of cache misses among the total number of requests.

We use the term *source* for the entity to which a client-cache sends requests. A source is defined by a set of one or more servers, the strategy they use to refresh their copies, and the pattern used by the client-cache to select a server. We are interested in how the type of the source affects the miss rate at the client-cache.

We assume objects always remain in the cache until they expire. In particular, there is no cache capacity constraint and no object can trigger an eviction of another, so the behavior of the cache on different objects is independent. Therefore, our analysis considers requests for a single object. Since we focus on age-induced effects, we generally consider sources that always provide a fresh, but possibly aged, copy of the object.

2.1 Source Types

In [2] we defined three types of *sources* which capture different relationships between a client-cache and its data source(s). The different sources are illustrated in Figure 1. The TTL value obtained through each source as a function of time is illustrated in Figure 2.

1. AUTH: An authoritative source. AUTH always provides a copy with zero age (TTL that equals the freshness lifetime).
2. EXC: Let α be drawn uniformly at random from the interval $[0, T]$ (the distribution $U[0, T]$), where T is the lifetime of the object. We call α the *displacement*. At time t an EXC source provides a copy whose age is $(t - \alpha) \bmod^3 T$ (TTL equals to $T - (t - \alpha) \bmod T$). Note that the choice of α is fixed for each "run" on the sequence, and the performance of EXC is the *expected* performance over all runs with different displacements.
   This source models a scenario where the client-cache upon each miss fetches a copy from the *same* high-level-cache. This high-level-cache maintains a fresh copy by refreshing it through an AUTH source each time it expires. The

[3] "mod" is a generalized modulu operation to arbitrary nonnegative numbers $a \bmod b = a - b * \lfloor a/b \rfloor$. If $b = 0$ then we define $a \bmod b \equiv 0$.

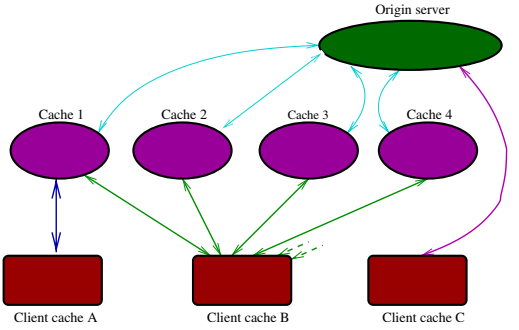


Fig. 1. Different types of sources: Cache A uses cache 1 as an EXC source. Cache B uses caches 1,2,... as an IND source. Cache C uses an AUTH source.

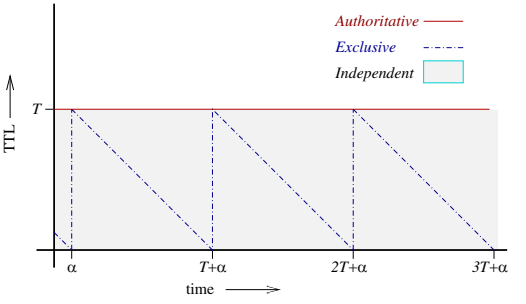


Fig. 2. TTL obtained through different types of sources. AUTH source always provides zero age and TTL of T. EXC source (shown with displacement α) provides age that cycles from 0 to T and thus TTL which cycles from T to 0. IND source provides at each point in time age drawn independently from $U[0,T]$ and thus TTL drawn from $U[0,T]$.

averaging over different displacements captures independence of the request times at the client-cache and the particular "refresh period" of the high-level-cache.

3. IND: Upon each miss at the client-cache, the IND source provides a copy with age independently drawn from $U[0,T]$ (thus a TTL that is also drawn from $U[0,T]$).
   This source models a scenario where upon each miss, the client-cache forwards the request to a different independent EXC-type high-level-cache. Independence means that the displacements of the different high-level-caches are not correlated.

The EXC, IND, and AUTH sources model pure scenarios. Currently, many Web caches direct requests for a particular URL through a high-level-cache (e.g., a proxy or a reverse proxy) and hybrids of the IND and EXC sources capture scenarios where several caches are used for load balancing purposes.

Our analysis assumes that the request sequence is independent of the performance of the client-cache (that is, future requests do not depend on which previous requests constituted a hit or a miss).

Two properties used later on are *consistency of a source* and *synchronization* of the relation between a cache and its source. A source is *consistent* if the age provided at any given time is consistent with past values. Formally, if at time t, the source provided an age of Φ then at time $t + \Delta$ the source provides the object with age of at most $\Phi + \Delta$. The AUTH and EXC sources are consistent. In general, we expect sources that model a single server to be consistent.

We say that a client-cache is *synchronized* with a source if whenever the client-cache contains a copy of the object which expires at some time t, then requests directed to the source at times $t + \Delta$ ($\Delta > 0$) obtain an object whose age is not more than Δ. By definition, a client-cache is always synchronized with AUTH and EXC sources but not with an IND source. As we shall see, synchronization helps performance. Intuitively, synchronization means that the copy at the source expires at the same time as the copy at the client-cache, and thus, misses at the client-cache on requests which closely follow previous requests are more likely to yield a copy with small age.

2.2 Extended Lifetime at a Client Cache

We consider situations where a client cache uses a longer lifetime duration than used by the source. As elaborated in the introduction, longer lifetimes can be the outcome of different settings or simply an attempt by the client-cache to reduce its miss-rate and decrease traffic between itself and its source. Lifetime extension trades increased miss-rate with increased staleness. We consider client-caches that use an extended lifetime value of $r * T$ for some fixed $r > 1$ (As before, T is the lifetime value as used by the source). We refer to r as the *extension factor*. We use the notation AUTH(r) (respectively, IND(r), EXC(r)) for a source of type AUTH (respectively, IND, EXC) in conjunction with a client-cache applying an extension factor of r.

2.3 Rejuvenating Sources

We consider replicating servers that refresh selected objects as soon as their TTL drops below some threshold (rather than wait for it to completely expire). We refer to such configured periodic pre-term refreshes as *rejuvenation*.

We extend the definitions of our basic sources to include rejuvenation. The source EXC_v is an EXC source that refreshes its copy of the object when the age exceeds v fraction of the lifetime value. Formally, let α be drawn from $U[0, vT]$. At time t, an EXC_v source return the object with age $(t - \alpha) \bmod (v * T)$ (so the TTL is $T - (t - \alpha) \bmod (v * T)$). As with an EXC source, α is fixed for a "run", and performance is the expected performance over runs with different displacements. We say that a client-cache uses an IND_v source if upon each miss it forward the request to a different independent EXC_v source. Hence, IND_v source returns copies with age drawn from $U[vT, T]$ and thus TTL drawn from $U[(1 - v)T, T]$. For both IND_v and EXC_v sources, a rejuvenation interval of $v = 1$ corresponds to the

respective pure source: $\text{EXC}_1 \equiv \text{EXC}$ and $\text{IND}_1 \equiv \text{IND}$. A rejuvenation interval of $v = 0$ corresponds to a pure AUTH source. That is, $\text{EXC}_0 \equiv \text{AUTH}$ and $\text{IND}_0 \equiv \text{AUTH}$.

Intuitively, we might expect a monotonic improvement in miss rate as v decreases from $v = 1$ to $v = 0$. We show later on that this is the case with IND_v sources but not with EXC_v sources. Note that a client-cache is synchronized with its EXC_v source if and only if $1/v$ is integral.

3 Relations between the Basic Sources

In this section we demonstrate some basic relations between the performance of a client cache through the different sources that we defined. We first prove that on any request sequence the miss rate of a client-cache working through an AUTH is never greater than through an EXC, which is in turn at most the miss rate through IND. To do this, we first establish two basic lemmas relating the miss rate at a client cache to the distribution of ages (and therefore TTLs) provided by the source. These lemmas will be also useful in later sections.

Lemma 1. *Consider two consistent sources s_1 and s_2 such that at any given point in time, the TTL available from s_1 is not smaller than the TTL available from s_2. Then, for any sequence of requests, the number of misses through s_1 is no larger than the number of misses through s_2.*

Proof. The proof is by induction on the length of the request sequence. The lemma clearly holds for sequences of length 1 and 2. Consider a request sequence τ of length l. If the last request is either a miss or a hit through both s_1 and s_2, or a miss through s_2 but a hit through s_1, then the lemma follows by using the induction hypothesis on the prefix of length $l - 1$ of τ.

Consider the case where the last request is a miss through s_1 but a hit through s_2. Let the ith request be the last request on which the client working through s_2 had a miss. The client working through s_1 must have had a hit on the ith request and all the requests following it including the next to last request. (Otherwise our assumptions guarantee that it would have got a TTL that covers the last request.) Thus by applying the induction hypothesis on the prefix of τ of length $i - 1$ the lemma follows.

The proof of the following lemma is given in the full version of the paper

Lemma 2. *Consider two sources s_1 and s_2 that serve objects with TTLs that are independently drawn from distributions with CDFs (Cumulative Distribution Functions) F_1 and F_2, respectively. Suppose that for all $x \geq 0$, $F_1(x) \leq F_2(x)$ (s_1 provides longer TTLs). Then for any sequence of requests, the expected number of misses through the source s_1 is no larger than the expected number of misses through the source s_2.*

An immediate consequence of Lemma 1 is that AUTH has smaller miss rate than other source types. The following Lemma establishes that EXC always outperforms IND.

Lemma 3. *For any sequence of requests, the miss rate through an* EXC *source is never higher than through an* IND*.*

Proof. Consider the ith request. Its likelihood of being a miss with EXC is $\min\{1, (t_i - t_{i-1})/T\}$. Let $m[j]$ and $h[j]$ denote the events that the jth request is a miss or a hit respectively, with an IND source. The likelihood that the ith request constitutes a miss with IND is

$$\begin{aligned} p(m[i]) &= p(m[i-1]) \min\{1, \frac{t_i - t_{i-1}}{T}\} \\ &\quad + p(h[i-1] \cap m[i-2]) \min\{1, \frac{t_i - t_{i-2}}{T}\} \\ &\quad + p(h[i-1] \cap h[i-2] \cap m[i-3]) \min\{1, \frac{t_i - t_{i-3}}{T}\} + \ldots \\ &\geq \min\{1, \frac{t_i - t_{i-1}}{T}\} \end{aligned}$$

Note that with respect to one fixed displacement value α, the EXC source could perform worse than IND. Consider the following sequence where requests are made at times $(2i + \epsilon)T$ and $(2i - \epsilon)T$ for integral $i > 0$. Suppose that the source refreshes at times iT. Then all requests would constitute misses with EXC, and only $1/2 + 2\epsilon$ would constitute misses with IND. Lemma 3 shows that on average over all displacements, EXC performs at least as well.

4 Rejuvenations and Extensions

In this section we show that there is a relation between rejuvenating sources (EXC_v sources) and clients extending the freshness lifetime of the object (EXC(r) sources). We state the following theorem (its proof will be given in the full version of the paper):

Theorem 1. *Consider the sources* EXC_v*, and* EXC*(r), for* $r = \frac{1}{v}$*. Then the miss rate of a client-cache through* EXC_v *on a request sequence* $\{t_i\}$ *is identical to the miss rate of a client-cache through* EXC*(r) on the sequence* $\{rt_i\}$*. The same is true for* IND *sources.*

Notice that Theorem 1 does not imply that extensions and rejuvenations are equivalent effects, as the miss rates are measured for different sequences. For example, applying this correspondence to two instances with the same sequence and sources EXC_v and EXC_u, we obtain two instances with two different request sequences (and two different extension factors).

4.1 Does Lifetime Extension Pay Off?

We first observe that generally, extending an object's lifetime cannot increase the miss rate. We then show that sometimes, however, extending the lifetime does not decrease the miss-rate so as a result we increase staleness without a corresponding gain.

Lemma 4. *Consider two client caches c_1 and c_2 that both use either an* AUTH, EXC, *or an* IND *sources. Consider a request sequence. If either of the following holds*

- *c_1 uses the same lifetime value as the source. c_2, upon some cache misses, uses an extended lifetime.*
- *The object has lifetime of T. c_1 uses lifetime of $r_1 * T$ and c_2 uses lifetime of $r_2 * T$ $(r_2 \geq r_1 \geq 1)$.*

Then the miss-rate of c_2 is at most that of c_1.

Proof. The proof for AUTH follows from Lemma 1. The proof for EXC with respect to any fixed displacement follows from Lemma 1, and still holds when we take the expectation over all displacements. The claim for IND source follows from Lemma 2.

The following Lemma shows that we may not reduce the miss rate for some frequently requested objects by increasing the extension factor to a non integral value. Therefore in such cases we increase staleness without a corresponding gain in miss rate.

Lemma 5. *Consider an* EXC(r) *source and a request sequence at the client cache such that the object is requested at least once every $(\lceil r \rceil - r)T$ time units. Then, the miss rate of the client-cache on such sequence is the same as with* EXC$(\lfloor r \rfloor)$,

Proof. When the object expires at the subsidiary, the copy at the parent has age $(r - \lfloor r \rfloor)T$. The object is requested before the parent refreshes its own copy. Thus, a miss at the client cache is incurred once every $\lfloor r \rfloor T$ time units.

For low request rates, however, non-integral extension factors could be beneficial even with EXC(r). For example, if an object is requested with intervals of $1.2T$, then the miss rate would be 1 with extension factor $r \leq 1.2$ and would be 0.8 with $r = 1.5$.

4.2 Does Rejuvenating Pay Off?

At first glance, it seems that rejuvenations can only improve performance at the client cache. Somewhat surprising is the fact that this is not the case for EXC rejuvenating sources. As a corollary of Theorem 1 and Lemma 5, we get the following lemma:

Lemma 6. *Let $v < 1$, and let $u = \frac{1}{\lfloor 1/v \rfloor}$. Consider a request sequence $\{t_i\}$, such that the object is requested at least once every $(\lceil \frac{1}{v} \rceil v - 1)T$ time units. Let m_v denote the miss rate through* EXC$_v$ *on $\{t_i\}$ and m_u the miss rate through* EXC$_u$ *on $\{t_i\}$. Then, $m_v = \frac{u}{v} m_u$.*

Proof. First, let $r = \frac{1}{v}$. By Theorem 1, m_v equals the miss rate through EXC(r) on $\{rt_i\}$. Since in $\{rt_i\}$ there is a request every $(\lceil \frac{1}{v} \rceil - \frac{1}{v})T = (\lceil r \rceil - r)T$ times units then using Lemma 5 we obtain that the miss rate of $\{rt_i\}$ through EXC(r)

is equal to its miss rate through EXC($\lfloor r \rfloor$). We claim that the miss rate through EXC($\lfloor r \rfloor$) on $\{rt_i\}$ equals $\frac{r}{\lfloor r \rfloor}$ times the miss rate through EXC($\lfloor r \rfloor$) on $\{\lfloor r \rfloor t_i\}$. Assuming this claim is established then by applying Theorem 1 again the theorem follows.

To prove the claim, notice that by the terms of the lemma, for every $i \geq 1$, $t_{i+1} - t_i < (\lceil \frac{1}{v} \rceil v - 1)T$, and thus, $rt_{i+1} - rt_i < (\lceil r \rceil - r)T < T$. Therefore, when using an EXC($\lfloor r \rfloor$) source, for both the sequence $\{rt_i\}$, and $\{\lfloor r \rfloor t_i\}$ we have a miss in the first T-interval after the extended lifetime of the last miss is over. Hence, we have a miss every $\lfloor r \rfloor T$ time units, and thus, the claim holds.

Lemma 6 demonstrates that the performance of a cache which receives frequent requests to the object can be strictly worse through EXC_v with $v < 1$ than through a non-rejuvenating EXC source. For example, for $v > 0.5$, and for sequences satisfying that the object is requested at least once every $(2v-1)T$ time units, by Lemma 6 the miss rate is strictly worse through EXC_v than with a non-rejuvenating EXC source. In this case, rejuvenating does not pay off.

In contrast, IND sources do exhibit monotonic dependence of the miss rate on v:

Lemma 7. *On any sequence of requests through an* IND_v*, the miss rate is non-increasing with* v*.*

The lemma follows from the following corollary of Lemma 2.

Corollary 1. *Let* $m1$ *be the expected miss rate at a client cache when it uses* IND *source. Suppose now that servers in the source cluster use rejuvenation and let* $m2$ *be the respective expected miss rate. We assume that rejuvenations are independent of the request sequence. That is, the TTL distribution through a uniformly sampled server is fixed over time. Then* $m1 \geq m2$*.*

Proof. The TTL distribution is $U[0,T]$ without rejuvenation. With rejuvenation, the TTL upon a miss is independently drawn from some other distribution $R[0,T]$. Clearly the CDFs of R and U satisfy that for every t, $F_R(t) \leq F_U(t)$ and Lemma 2 applies.

The following corollary of Lemma 1 shows that although generally rejuvenation does not always improve the performance, rejuvenation cannot degrade performance on any sequence if the source preserves synchronization.

Corollary 2. *Suppose a rejuvenating* EXC *source adheres to the original refresh schedule, refreshing the object at times* $\alpha + iT$ *for integral* i *in addition to possibly rejuvenating it at other points in time. Then on any sequence of requests, the number of misses is not higher than the number of misses through* EXC*.*

Proof. Follow-up rejuvenations guarantee that at any given point in time, the TTL obtained from the source with rejuvenation is at least as large as the TTL obtained without rejuvenation. The proof follows by Lemma 1.

In particular, performance through EXC_v with integral $1/v$, (i.e., $v = 1/2, 1/3, \ldots$), is at least as good as through EXC. In the full version of the paper we show that even if we restrict ourselves to EXC_v, with integral $1/v$, the miss rate is not always monotonic as a function of v, and moreover, there are examples where EXC_v performs worse than an IND source.

5 Worst Case Performance

In this section we analyze the worst case performance of EXC and IND rejuvenating sources. For a source S and a request sequence $\{t_i\}$ let $\rho(\{t_i\}, S)$ be the ratio between the expected miss rate of $\{t_i\}$ through S and through an AUTH source. The supremum over all sequences $\{t_i\}$ of $\rho(\{t_i\}, S)$ is called the competitive ratio of S, and is denoted by $\rho(S)$. Let P be a consistent source that may decide occasionally to rejuvinate the object based on requests that it obtained so far. We call such a source a *rejuvinating policy*. We first analyze the miss rate of rejuvinating policies. Notice that for any given v, EXC_v is a rejuvinating policy (that simply takes decisions which are independent of the sequence). We next show that the competitive ratio of any rejuvinating policy is no greater than 2.

Theorem 2. *The competitive ratio of any rejuvinating policy is at most 2.*

Proof. We first establish that the client can get at most two misses in each time interval of length T. Consider two consecutive misses. Since the rejuvinating policy is consistent the sum of the TTL obtained at the second miss, and the elapsed time between the two misses must be at least T. Thus, a subsequent miss can occur only after at least T time units past the first miss. We now show that the miss rate of every policy is at most twice the optimum. Let the miss-times through an AUTH source be $m_1, m_2, \ldots, m_k$. Clearly, the set of all request times is covered by the mutually disjoint intervals $[m_i, m_i + T)$, as a request which is not covered by any of these intervals must be a miss. In each of these intervals, the AUTH source has exactly one miss, while the rejuvinating policy could have had at most 2, and thus the theorem holds.

Next we establish a corresponding lower bound of 2 on the performance of any rejuvinating policy that cannot rejuvinate the object continously but must have a small gap of ϵ between consecutive rejuvinations.

Theorem 3. *Consider a randomized rejuvenating policy P on a consistent source such that the time interval between any two rejuvenation points must exceed some $\epsilon > 0$. Then P has competitive ratio $\rho(P) = 2$.*

Proof. By Lemma 1 we get that $\rho(P) \leq 2$. Let $\delta > 0$, and we prove that $\rho(P) \geq 2 - \delta$. We use Yao's lemma [8], and construct a distribution on sequences of requests $r_1, r_2, \ldots$ at times $t_1, t_2, \ldots$, such that the miss rate of the optimal policy is always $\frac{1}{2}$, while the expected miss rate of P is $1 - \delta/2$.

Let t_1 be chosen uniformly from an arbitrary interval of length ϵ. For every $k \geq 1$, let $t_{2k} = t_{2k-1} + T - \epsilon\delta$, and let t_{2k+1} be chosen from $U[t_{2k} + 2T, t_{2k} + 2T + \epsilon)$. Using the property of P, in which the smallest distance between any two rejuvenation points is ϵ, it is easy to see that for every $k \geq 1$, the probability over the request times distribution, that the TTL of r_{2k-1} when using P is at least $T - \epsilon\delta$, is at most δ. Moreover, r_{2k-1} constitutes a miss under any policy, including P. The request r_{2k} is a miss of P with probability at least $1 - \delta$. Thus, the expected miss rate of P is at least $(2 - \delta)/2$. The optimal policy can rejuvenate just before the client incurs a miss, hence has a miss rate of $1/2$. Therefore, using Yao's lemma [8], $\rho(P) \geq 2 - \delta$, and the theorem holds.

In particular, Theorem 2 implies that the source EXC_v with any $v > 0$ has a competitive ratio 2. A related problem is the competitive ratio of IND_v. Since an IND_v source is not consistent it is not a rejuvinating policy and therefore Theorem 3 does not apply. To give some intuition, if $v < 1/2$, then the TTL provided for every request is at least $T/2$, and thus, in every interval of length T time units, there are at most 2 misses. Therefore, in this case, the proof for Theorem 2 holds, and thus, the competitive ratio of IND_v is at most 2. To see that it is exactly 2, consider a sequence where requests are spaced T time units apart. The miss rate is 1 through IND_v source whereas it is $1/2$ through an AUTH source. We obtain the same ratio of 2 on a sequence of requests that is constituted of subsequences, where subsequences are spaced more than T apart and in each subsequence the distance between the last request and the first request is exactly T. More generally, we prove the following theorem:

Theorem 4. *The competitive ratio of* IND_v *is* $\sum_{k=0}^{\lfloor \frac{1}{1-v} \rfloor} \frac{(1-k(1-v))^k}{k! v^k}$.

Thus, by Theorem 4, for $v < 1/2$, the competitive ratio of an IND_v source is 2, and as v approaches 1, the competitive ratio approaches $e = 2.718....$ Interestingly, this worst-case analysis supports the relation between basic sources established in Section 3: EXC source has competitive ratio of 2 whereas IND source has a strictly worse competitive ratio of e. The proof of Theorem 4 is provided in the full version and is based on establishing that the competitive ratio of IND_v equals the expectation of Y_v, where Y_v is defined as follows. Let $X_1, X_2, \ldots$ be a sequence of independent random variables drawn from $U[T(1-v), T]$. Define Y_v as the random variable, such that $Y_v = k$ if and only if $X_1 + \ldots + X_{k-1} \leq T < X_1 + \ldots + X_k$.

6 Conclusion and Open Problems

We modeled and analyzed age-related performance issues for a distributed system of caches that use TTL-based consistency. Our analysis revealed interesting patterns that may seem counter-intuitive. Despite the wide-scale use of TTL-based consistency (for Web caches and within the Domain Name System), its performance effects are not properly understood by practitioners [3]. The models we used closely follow the true behavior of Web caches and we believe that our results provide insights for increasing the effectiveness of Web caches.

Our results are complemented and supported by extensive simulations using Pareto arrivals and traces from Web content caches and analysis of request sequences with Poisson and fixed inter-request-time distributions (see [2] and the full version). In particular, these results further demonstrate the presence of interesting dependencies of performance on the extension factor or rejuvenation interval values. Interestingly, the patterns are similar for the heavy-tailed Pareto, Poisson, and for actual request sequences, but yet, are not universal (e.g., does not occur with fixed inter-arrival times). An intriguing open question is thus to characterize the class of request distributions for which these patterns occur.

References

1. T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol — HTTP/1.0. RFC 1945, MIT/LCS, May 1996.
2. E. Cohen and H. Kaplan. Aging through cascaded caches: performance issues in the distribution of Web content. Manuscript, 2000.
3. E. Cohen and H. Kaplan. The age penalty and its effect on cache performance. In *Proc. of the 3rd USENIX Symposium on Internet Technologies and Systems*, 2001.
4. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, and T. Leach, P. Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. RFC 2616, ISI, June 1999.
5. Inktomi Traffic Server. `http://www.inktomi.com`.
6. A Distributed Testbed for National Information Provisioning. `www.ircache.net`.
7. Squid internet object cache. `http://squid.nlanr.net/Squid`.
8. A. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 17th Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977.

Routing in Trees

Pierre Fraigniaud* and Cyril Gavoille

[1] CNRS, Laboratoire de Recherche en Informatique,
Université Paris-Sud, 91405 Orsay cedex, France.
pierre@lri.fr
[2] Laboratoire Bordelais de Recherche en Informatique, Université Bordeaux I, 33405 Talence Cedex, France.
gavoille@labri.fr

Abstract. This article focuses on routing messages along shortest paths in tree networks, using compact distributed data structures. We mainly prove that n-node trees support routing schemes with message headers, node addresses, and local memory space of size $O(\log n)$ bits, and such that every local routing decision is taken in constant time. This improves the best known routing scheme by a factor of $O(\log n)$ in term of both memory requirements and routing time. Our routing scheme requires headers and addresses of size slightly larger than $\log n$, motivated by an inherent trade-off between address-size and memory space, i.e., any routing scheme with addresses on $\log n$ bits requires $\Omega(\sqrt{n})$ bits of local memory-space. This shows that a little variation of the address size, e.g., by an additive $O(\log n)$ bits factor, has a significant impact on the local memory space.

Keywords: compact routing, trees, routing algorithms.

1 Statement of the Problem

Delivering messages between pairs of processors is a basic and primary activity of any distributed communication network. This task is performed using a *routing scheme*, that is a mechanism working in a distributed fashion, and allowing source-to-destination messages delivery in the network, for any source and any destination.

Quite often, the design and the optimization of the management and control systems of the network, including routing, is done after the network construction. In this case, the routing problem is stated as follows: given a graph (i.e., the underlying topology of a communication network), design for each node of the graph (i.e., for each router of the network) a efficient computable function that determines, for every message entering a node, the link on which the message has to be forwarded. This computation is performed as function of the information contained in the header of the message, e.g., the destination node address, and

* Part of this work was done while the first author was visiting Carleton University at Ottawa, and Concordia University at Montréal. Additional support from Carleton U, the NATO, and the RNRT project R.O.M.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 757–772, 2001.

of local information stored at the node. The term "efficient" groups a set of desirable quality factors like: routes of reasonable length (compared to shortest paths); low local computation time; limited link-congestion; compact local data structures; fault tolerance; etc.

We want to point out that the complexity results on routing are quite strongly dependent of the model, including, e.g., the ability to setup the addresses of the nodes, the ability to modify the headers, etc. Moreover, we also want to stress that results strongly differ according to slight variations of parameters, including, e.g., the size of the header, the length of the routes, etc. Therefore, let us make clear the framework of this paper.

In the remaining, the communication network is denoted by a connected and undirected n-node graph $G = (V, E)$. The routing scheme makes use of *addresses*, every node being identified by an address which is unique in the network. Hereafter, the address of node u is denoted by $\ell(u)$. The routing scheme makes also use of a *routing algorithm* that is a distributed algorithm whose role is to route messages. Upon reception of a message by node u, the routing decision taken at u is performed by a *routing function*. It depends solely on (1) the message header, (2) the incoming link, and (3) local data structures stored at u. The design of the routing scheme includes not only the design of the routing algorithm via the definition of the routing function, but also the setting of the address-set, the header-set, and of the local data-structures. Moreover, it has to incorporate the link labeling.

More precisely, incoming and outgoing links from every node u are identified by numbers from 1 to $\deg(u)$, the degree of u. This labeling is local, and should not necessarily be consistent between neighbors. In particular, the edge $\{u, v\}$ can be labeled i by u, and j by v, with $i \neq j$. Each node (i.e., each router) is also supposed to have an extra link, connecting it to its local host. This link is labeled 0. Once the routing function has selected the outgoing link on which the message must be forwarded, i.e., a number between 0 and $\deg(u)$, it creates a new header for that message, and it sends the message with its header through that link. The original header of a message is given by the source host; it simply consists of the address of the destination, and, upon reception of a message from the source host, the attached router may modify the header, but the result of this modification depends solely on the destination address, and, again, local data structures stored by the router. We point out here that no time or space consuming computations are hidden in this model.

A specific type of routing is called *direct*. Direct routing imposes that the message header is fixed at once by the source host, and cannot be modified by intermediate nodes on its route toward the destination. In other words, the header set is exactly the node address set. Direct routing schemes are desirable because they do not require extra hardware facilities to rewrite headers. Note though that relaxing this constraint allows to construct sophisticate routing schemes for arbitrary graphs (cf., e.g., the hierarchical routing schemes introduced in [1,10]).

To summarize, solving the routing problem consists in designing the whole routing scheme, including (1) the routing function, (2) the format of the ad-

dresses, (3) the local labeling of the links, (4) the format of the headers, and (5) the local information to be stored locally at each node in order to perform the computation of the routing function. The complexity of the setting-up of these data is measured by the time it takes to perform those computations from given the graph G, and is refered as the *pre-processing time*. The other parameters that are typically addressed in the literature are the following. The *size of the addresses* (that is also the *size of the headers* for direct routing schemes) is measured by the range of the set in which these addresses are taken. More precisely, for $m \geqslant n$, a routing scheme is of $[1, m]$-range address if the addresses are taken from the set $\{1, 2, \ldots, m\}$. (Alternatively, such a scheme is called a $\log m$-bit address routing scheme.) The *local memory space* is the size of all the data structures required by each node to compute the routing function. The *routing time* is the worst-case time-complexity of the routing function to perform its computation. The *length of the routes* is measured by competition against the shortest paths.

In this paper we consider shortest-path routing only. Moreover, all the routing schemes we will construct are direct, i.e., routing will be performed based on the destination address only, and headers are not modified once set by the source. Regarding the local labeling of the links, we will consider two main variations of the problem: the *designer-port* model, and the *fixed-port* model. The former allows the designer of the routing scheme to freely label (between 1 and $\deg(u)$) the incident links of each node u while constructing and optimizing the data structures of the routing scheme. On the contrary, the latter assumes that the local labeling of the links is given is advance. The fixed-port model forces the use of correspondence tables if the routing function needs a specific labeling to perform efficiently, and therefore this model may lead to a significant increase of the size of the data structures compared to the designer-port model. On the positive side though, the fixed-port model allows to superimpose several routing schemes such as, e.g., hierarchical routing schemes [1,10].

Before stating the results included in this paper, we list some previous and related works.

Related Works

One of the most widely used routing scheme for arbitrary graphs is the *routing tables*. It consists in labeling the nodes from 1 to n, and to store an n-entry table at each node, such that its i-th entry returns the local number of the adjacent link on which a message of destination address i has to be transmitted. Routing tables are direct and use $[1, n]$-range addresses, require $O(n \log d)$ bits of memory at each node of degree d in the fixed-port model, and perform routing in $O(1)$ time at each node. Although $O(n \log d)$ memory bits seems quite high, it must be pointed out that this is the best that can be achieved for arbitrary graph. More precisely, it has been proved in [5] that $\Omega(n \log d)$ bits is a lower bound for shortest-path routing schemes using addresses on $c \log n$ bits, for every constant $c \geqslant 1$. This lower bound holds in the designer-port model. According to this

result, a tremendous effort has been done to derive routing schemes requiring smaller memory space for specific classes of networks, in particular for trees.

In trees, a common strategy consists in setting the addresses via a Depth-First Search (DFS) traversal of the tree, and then to use the *interval routing scheme* (IRS) of [11,13] (see [4] for a survey on IRS). More precisely, at every internal node u of the tree, the edge leading to a child v is associated to an interval containing the set of the addresses of the nodes in the subtree rooted at v, and the link leading to the root is associated to the complementary interval (this latter interval may wrap modulo n). The memory space is $O(d \log n)$ bits at each node of degree d in the fixed-port model, and the use of a binary search in the set of intervals provides a $O(\log n)$ routing time at each node. The total amount of routing informations, summed over all nodes, is low compared to the routing tables, i.e., $O(n \log n)$ bits in total. However, IRS does not evenly balance routing information among the nodes since $O(n \log n)$ bits could be required locally for trees of large degree.

Cowen [2] proposed a direct routing scheme in the fixed-port model for trees of large degree using $O(\sqrt{n} \log n)$ bits of memory, $3 \log n$ bits of addresses, and $O(\log n)$ routing time. The address-space can be optimized considering the designer-port model: the direct routing scheme in [3] uses $O(\sqrt{n} \log n)$ bits of memory with $[1, n]$-range addresses, and $O(\log n)$ routing time. Actually, if we insist on small memory space, [4] shows that there exists a routing scheme for trees with $[1, n]$-range addresses, $\alpha\sqrt{n}$ bits of memory, where $\alpha \approx 3.7$. However, this latter scheme may involve exponential routing times.

Peleg [9] proposed the best known routing scheme for trees, inspired from the so-called "distance labeling" schemes, and stated in the fixed-port model. It is based on a recursive decomposition by "center" of the tree, using $O(\log n)$ levels. It uses $O(\log^2 n)$ bits for both the addresses and the memory at each node, and requires $O(\log n)$ routing time. Actually, the routing algorithm is direct and based on the destination-address only so that the node needs only to store its address. That type of routing schemes are called "memory-free" routing or "label-based" routing. As it is remarked in [9], every direct routing scheme can be transformed into a memory-free routing scheme by extending the size of each address so that it contains all the local data structures of the node.

This paper improves all these previous routing schemes for trees, including the one in [9].

Our Contributions

1. We propose a direct shortest-path routing scheme for trees asymptotically using $5 \log n$ bits for the addresses, $3 \log n$ bits of memory space, and $O(1)$ routing time[1]. This scheme is asymptotically optimal for all of the considered measures: memory space, header size, and routing time. Reducing

[1] All the time complexities are computed in a word-RAM model where standard integer operations on words of $O(\log n)$ bits are performed in constant time, including logical bitwise operations, integer division, and integer multiplication. All the logarithms are in base 2.

the address range can be done, but up to an increase of the routing time. More precisely, we also propose a direct shortest-path routing scheme using $2.8 \log n$ bits for the addresses, $3 \log n$ bits of memory space, but involving polynomial time routing decisions.

Attaching the address with the local data structure at each node, we thence show that there exists (1) a shortest-path memory-free routing scheme for trees with constant routing time and $7 \log n$ bits of addresses, and (2) a shortest-path memory-free routing scheme for trees with a $4.8 \log n$ bits of addresses and polynomial routing time. All these results are stated in the designer-port model. A similar result was founded independently by Thorup and Zwick [12].

2. For the fixed-port model, we propose a direct shortest-path routing scheme using $O(\log^2 n / \log \log n)$ bits for the addresses, $O(\log^2 n / \log \log n)$ bits of memory space, and with constant routing time. We obtain a shortest-path memory-free routing scheme using $O(\log^2 n / \log \log n)$ bits with constant routing time. These results improve the best known result by a factor $O(\log \log n)$.
3. We also show lower bounds. In particular, we show that, in the designer-port, model any shortest-path routing scheme for arbitrary tree requires a memory space of at least $\alpha\sqrt{n} - o(\sqrt{n})$ bits if using $[1, n + o(\sqrt{n}/\log n)]$-range addresses. This improves and generalizes the lower bound of [3], and shows the optimality of the results therein. By comparing this lower bound with our first contribution, it also shows that a little variation of the address size, e.g., by an additive $O(\log n)$ bits factor, has a significant impact on the local memory space.

   For the fixed-port model, we prove a $n - o(n)$ bits lower bound on the memory space for shortest-path routing scheme on trees using $[1, n + o(n/\log n)]$-range addresses. In contrast with this bound, we construct a direct shortest-path routing scheme with $[1, n]$-range addresses, using at most $n + o(n)$ bits of memory space, and constant routing time. This demonstrates an optimal tradeoff between address-size and memory space.

2 A Routing Scheme with $O(\log n)$ Bits of Address and Memory Space

Let T be any n-node tree, and let r be any node of T. From now, T is supposed to be rooted at r. For every node u, let us denote by T_u the subtree of T rooted at u induced by u and all its descendents. We denote by $w(u)$ the number of nodes of T_u, and we call it the *weight* of u. We define $\mathrm{id}(u)$ as the numbering of the nodes of T by consecutive integers in $[1, n]$ obtained by a DFS traversal of T performed under the following guideline: at each internal node, the DFS numbering visits first the largest subtree, then the second largest subtree, and so on. It yields a labeling in which $\mathrm{id}(r) = 1$, and, for every internal node u and every two of its children x, y, if $\mathrm{id}(x) \leqslant \mathrm{id}(y)$ then $w(x) \geqslant w(y)$.

2.1 Port and Address Assignment

If y is a child of x in T, then we define the *rank* of x, denoted by $\text{rank}(x, y)$, as the number of children of x whose DFS labels are at most $\text{id}(y)$. The labeling of the edge $\{u, v\}$ at u is denoted by $\text{port}(u, v)$. We set $\text{port}(r, v) = \text{rank}(r, v)$ for any child v of the root r, and, for any $u \neq r$,

$$\text{port}(u, v) \;=\; \begin{cases} 1 & \text{if } v \text{ is the parent of } u; \\ 1 + \text{rank}(u, v) & \text{otherwise.} \end{cases}$$

The extra edge linking any node u to its local host is labeled 0. For every node $u \neq r$, we define $\text{path}(u)$ as the sequence of ranks encountered on the path from r to u. (We set $\text{path}(r) = ()$, the empty sequence.) More precisely, if the path from r to u is the sequence of nodes $r = u_0, u_1, \ldots, u_t = u$ then $\text{path}(u) = (\text{rank}(u_0, u_1), \ldots, \text{rank}(u_{t-1}, u_t))$. We define the *clean-path* of u as the sequence $\text{cpath}(u)$ obtained from $\text{path}(u)$ by removing all the ranks that are equal to 1. We denote by $|\text{cpath}(u)|$ the length of the sequence $\text{cpath}(u)$, i.e., the number of ranks of $\text{path}(u)$ distinct from 1. In particular, $|\text{cpath}(u)| = 0$ if $\text{path}(u)$ consists of 1's only.

For every node u, the address $\ell(u)$ of u satisfies $\ell(u) = \langle \text{id}(u), \text{cpath}(u) \rangle$.

The data structure stored at node u is

$$\text{table}(u) \;=\; \langle \text{id}(u), w(u), w_1(u), |\text{cpath}(u)| \rangle$$

where $w_1(u)$ is the weight of the heaviest child of u, i.e., the weight of $w(z)$ such that $\text{rank}(u, z) = 1$. For a leaf u, we set $w_1(u) = 0$. The next lemma will be useful for computing the space needed for storing $\text{table}(u)$.

Lemma 1.
(1) If u is the parent of v, then $\text{rank}(u, v) \leqslant w(u)/w(v)$.
(2) If $\text{cpath}(v) = (p_1, \ldots, p_k)$, *then* $\prod_{i=1}^{k} p_i \leqslant n/w(v)$, *and* $k \leqslant \log(n/w(v))$.

Proof.

(1) By definition, if $\text{rank}(u, v) = i$ then there are i children of u, say $c_1, \ldots, c_{i-1}, c_i = v$, such that $\text{id}(c_j) \leqslant \text{id}(v)$. For every such node c_j, $w(c_j) \geqslant w(v)$. Thus, $\sum_{j=1}^{i} w(c_j) \geqslant i \cdot w(v)$. Since $w(u) \geqslant \sum_{j=1}^{i} w(c_j)$, it follows that $w(u) \geqslant i \cdot w(v)$ that is $\text{rank}(u, v) \leqslant w(u)/w(v)$ as claimed.

(2) By construction, $p_i = \text{rank}(u_i, v_i)$ for some edge (u_i, v_i), u_i parent of v_i. By (1), $p_i \leqslant w(u_i)/w(v_i) \leqslant w(u_i)/w(u_{i+1})$ since $w(u_{i+1}) \leqslant w(v_i)$. It follows that $\prod_{i=1}^{k} p_i \leqslant w(u_1)/w(v_k)$. We have $w(u_1) \leqslant n$, and $v_k = v$. Since $p_i \geqslant 2$, $k \leqslant \log\left(\prod_{i=1}^{k} p_i\right) \leqslant \log(n/w(v))$. ∎

2.2 Routing Algorithm

Our routing algorithm is direct, i.e., a message-header consists solely of the destination address of that message, and it will not be modified along its path

from its source to its destination. Assume that a node x receives a message of header $h = \ell(y)$. The routing decision at x is described by a function $\textsc{Route1}(x, h) \in [0, \deg(x)]$ that returns the local label of the link on which the message has to be forwarded from x. In the following, let b denote the value 0 if $\mathrm{id}(x) = 1$ (i.e., if $x = r$), and 1 otherwise.

$$\textsc{Route1}(x, \ell(y)) = \begin{cases} 0 & \text{if } \mathrm{id}(y) = \mathrm{id}(x); \\ 1 & \text{if } \mathrm{id}(y) < \mathrm{id}(x) \text{ or } \mathrm{id}(y) \geqslant \mathrm{id}(x) + w(x); \\ 1+b & \text{if } \mathrm{id}(x) < \mathrm{id}(y) \leqslant \mathrm{id}(x) + w_1(x); \\ p+b & \text{otherwise, where } p = (|\mathrm{cpath}(x)|+1)\text{-th element of } \mathrm{cpath}(y). \end{cases}$$

The next section shows that the routing algorithm is correct, and section 2.4 shows how to implement the function Route1.

2.3 Correctness

Lemma 2. *For every pair of nodes x, y, the routing algorithm described by* Route1 *routes any message from x to y along a shortest path.*

Proof.

If $\mathrm{id}(y) = \mathrm{id}(x)$, then the function returns the correct answer since the message is arrived at destination.

If $\mathrm{id}(y) < \mathrm{id}(x)$ or $\mathrm{id}(y) \geqslant \mathrm{id}(x) + w(x)$, then, by definition of the DFS numbering, y is not a descendent of x, and thus the message has to be transmitted to the parent of x. The routing function returns in this case the correct answer, i.e., port 1.

If $\mathrm{id}(x) < \mathrm{id}(y) \leqslant \mathrm{id}(x) + w_1(x)$ then y is a descendent of x. Since $\mathrm{id}(y) \leqslant \mathrm{id}(x)+w_1(x)$, the message has to be transmitted on the port of the heaviest child of x because the DFS numbering labels first the largest subtree of x. Therefore, the routing function correctly answers port $1 + b$ (i.e., 1 if $\mathrm{id}(x) = 1$, and 2 otherwise), because links are locally labeled in a decreasing order of the weights of the subtrees.

If $\mathrm{id}(x) + w_1(x) < \mathrm{id}(y) < \mathrm{id}(x) + w(x)$ then y is a descendent of x and the sequence $\mathrm{path}(x)$ is a prefix of $\mathrm{path}(y)$. Therefore, $\mathrm{cpath}(x)$ is also a prefix of $\mathrm{cpath}(y)$. The message has to be transmitted to a child of x, say z. Note that $\mathrm{path}(z) = (\mathrm{path}(x), \mathrm{rank}(x, z))$, and note also that $\mathrm{rank}(x, z) \neq 1$ because z is not the heaviest child of x since $\mathrm{id}(x) + w_1(x) < \mathrm{id}(y)$. Since $\mathrm{rank}(x, z) \neq 1$, and since $\mathrm{cpath}(x)$ is a prefix of $\mathrm{cpath}(y)$, we have that $\mathrm{rank}(x, z)$ is contained in the sequence $\mathrm{cpath}(y)$. The value $\mathrm{rank}(x, z)$ is precisely the first element of $\mathrm{cpath}(y)$ located after all the elements in common with $\mathrm{cpath}(x)$. Hence, it is the $(i + 1)$-th element of $\mathrm{cpath}(y)$, where i is the number of elements of $\mathrm{cpath}(x)$. Thus, the routing function returns the correct answer in this case as well, completing the proof. ∎

2.4 Implementation

Lemma 3. *For every node u, its address $\ell(u)$ is of length less than $5\log n + 1$ bits, and its local data structure* table(u) *is of length less than $3\log n + \log\log n + 4$ bits.*

Proof. Recall that $\ell(u) = \langle \mathrm{id}(u), \mathrm{cpath}(u)\rangle$. We represent $\mathrm{id}(u)$ by a binary string of $\lceil \log n \rceil < (\log n) + 1$ bits. Now, let us give a compact representation of the sequence $\mathrm{cpath}(u) = (p_1, \ldots, p_k)$. Let l_i be the length of the binary decomposition of $q_i = p_i - 2$ ($q_i \geqslant 0$ since $p_i \geqslant 2$). Precisely, $l_i = \max\{\lceil \log(p_i - 1)\rceil, 1\}$. The sequence $\mathrm{cpath}(u)$ is represented by two binary strings, say A and B, each of length $L = \sum_{i=1}^{k} l_i$, as follows: A is composed of the concatenation of the binary representation of the q_i's on l_i bits; and B is a bitmap that indicates the begining of each field representing a q_i in A (this is doable since $l_i \geqslant 1$).

Note that $p_i \geqslant 2$ implies $l_i < (\log p_i) + 1$. Hence, $L < \log\left(\prod_{i=1}^{k} p_i\right) + k$. By Lemma 1, $L < \log(n/w(u)) + \log(n/w(u)) \leqslant 2\log n$. In total, $\mathrm{id}(u)$, A, and B can be represented by less than $5\log n + 1$ bits.

Finally, table$(u) = \langle \mathrm{id}(u), w(u), w_1(u), k\rangle$, where $k = |\mathrm{cpath}(u)|$, can be represented by a binary string of length at most $3\lceil \log n\rceil + \lceil \log k\rceil < 3\log n + \log\log n + 4$ bits (by Lemma 1, $k \leqslant \log n$). ∎

Actually, $\ell(u)$ can be represented in a more compact way. Indeed, the coding of $\mathrm{cpath}(u)$ is not the best possible. To improve the coding we need the following lemma originally proved by Kalmár in 1930, and that can be also founded in [14].

Lemma 4. [14] *Let $Z(n)$ be the number of ordered integer sequences $p_1, p_2, \ldots$ such that $p_i \geqslant 2$ and $\prod_{i\geqslant 1} p_i \leqslant n$. Then, $Z(n) \sim n^{\rho}/\kappa$, with $\kappa = -\rho\cdot\zeta'(\rho)$ where ρ is the unique real solution of the equation $\zeta(\rho) = 2$ and ζ is the Riemann Zeta function. (We have $\rho \approx 1.7286$ and $\kappa \approx 3.1429$.)*

From this lemma, every clean-path can be coded using at most $\lceil \log Z(n)\rceil \leqslant \rho\log n < 1.8\log n$ bits. Moreover, there exists a polynomial time algorithm that generates all such sequences[2]. Thus coding and decoding of $\mathrm{cpath}(u)$ can be done in polynomial time.

Note that in order to execute Route1$(x, \ell(y))$ we need to extract rapidly each field of $\ell(y)$ and of table(x). Given a binary string S of length L, let $\mathrm{ones}_S(i)$ denote the function that returns the number of 1-entries in S up to (and including) position i, and let $\mathrm{select}_S(j)$ denote the function that returns the position of the j-th 1-entry in S.

Lemma 5. [7,8] *Let S be a binary string of length L function of n, such that $L \leqslant n$. The operations* $\mathrm{ones}_S(\cdot)$ *and* $\mathrm{select}_S(\cdot)$ *can be performed in $O(1)$ time with $o(L)$ pre-computed auxiliary bits, thus using a total of $L + o(L)$ space. Moreover, the auxiliary bits can be computed in $O(L)$ time.*

[2] For instance one can generate all the $\lfloor 4\log n\rfloor$ bit strings and test if it is a valid coding into the set of A, B strings of clean-paths.

Theorem 1. *In the designer-port model, every n-node tree supports a direct shortest-path routing scheme with $5\log n+o(\log n)$ bits of address (resp., $2.8\log n$ bits), $3\log n+O(\log\log n)$ bits of memory space, and $O(1)$ (resp., $n^{O(1)}$) routing time. Moreover, the pre-processing time is $O(n\log n)$ (resp. $n^{O(1)}$).*

Proof. The address of a node u is composed of three binary strings: $\mathrm{id}(u)$, and A and B to represents $\mathrm{cpath}(u)$. We enhance the address of u with $o(\log n)$ bits in order to support the operation $\mathrm{select}_B(\cdot)$ in constant time by Lemma 5. The data structure $\mathrm{table}(u)$ is not modified.

Let us show how to execute $\textsc{Route1}(x,\ell(y))$ in constant time, for any pair of nodes x, y. The three first conditions involve constant-time computations. (Note that extracting fields of fixed length and at a fixed position can easily be done by the use of binary masks and shifting.) Actually, only Condition 4 needs more attention due to the extraction of the $(i+1)$-th element of $\mathrm{cpath}(y)$. If $\mathrm{cpath}(y)$ is compacted in $\lceil\log Z(n)\rceil$ bits, that is by storing an index of the sequence $\mathrm{cpath}(y)$ in the set of all such sequences, it suffices to decode $\mathrm{cpath}(y)$ from its index, and to extract the $(i+1)$-th element. This takes $n^{O(1)}$ time. However, if $\mathrm{cpath}(y)$ is represented by two binary strings A and B, one can extract the $(i+1)$-th element in $O(1)$ time. Its value p is located in A between the position $j_1 = \mathrm{select}_B(i+1)$ and $j_2 = \mathrm{select}_B(i+2)$ (excluded). Both indices can be computed in constant time. Thanks to a mask $M = (2^{j_1+1}-1)-(2^{j_2+1}-1)$ applied on the string A, and at the price of a simple shift, one can extract p (actually, $p-2$) from A, and thus compute correctly $\textsc{Route1}(x,\ell(y))$.

Computing the weights requires $O(n)$ time, and sorting them at each node takes $O(n\log n)$. The time to perform the DFS numbering is then $O(n)$, and the time to compute all the clean-path sequences is then $O(n\log n)$ since every clean-path is on $O(\log n)$ bits (resp., $n^{O(1)}$ time if clean-paths are represented on $\lceil\log Z(n)\rceil$ bits). The time to compute all the addresses and data structures is again $O(n\log n)$, including the generation of auxiliary tables for each node. Therefore, the total pre-processing time is $O(n\log n)$ (resp. $n^{O(1)}$ if compacted). ∎

Concatenating the address and the local data structure of each node, we obtain (recall that $\ell(u)$ and $\mathrm{table}(u)$ have $\mathrm{id}(u)$ in common):

Corollary 1. *In the designer-port model, every n-node tree supports a memory-free shortest-path routing scheme with $7\log n+o(\log n)$ bit for the addresses (resp. $4.8\log n$ bits), and $O(1)$ (resp. $n^{O(1)}$) routing time. Moreover, the pre-processing time is $O(n\log n)$ (resp. $n^{O(1)}$) time.*

3 Routing Schemes for the Fixed-Port Model

As mentioned in the introduction, on one hand, the fixed-port model is potentially less efficient than the designer-port model in the sense that it may require the uses of larger data structures, but on the other hand it allows the combination of several routing schemes. This latter property makes the fixed-port model worth to be studied.

We reuse all the notations introduced in Section 2. In particular, for u parent of v, the fixed-port model assumes that $\text{port}(u,v) \in [1, \deg(u)]$ is fixed, and cannot be modified. This has a significant impact on the size of the local data structures. Let us try to explain why. Assume that the routing function deals with colored messages. Under the designer-port model, a compact routing scheme could consist in labeling locally the links with colors so that red messages are simply routed along the red link. Under the fixed-port model, this is not possible, and red messages may need to be routed along a green link. Note that one cannot simply recolor the red messages in green because there might be other nodes for which the routing of green messages must be performed through the blue link. Actually, the computation of "green" from "red" has an inherent cost, and requires additional memory bits. There are mostly two solutions for the storage of these bits. One consists in storing them locally, e.g., node u stores a "correspondence table" of $O(d \log d)$ bits, where $d = \deg(u)$. Another consists in storing them in the addresses of the nodes, e.g., all nodes w requiring the use of the link labeled $\text{port}(u,v)$ to route messages from u to w have addresses containing additional bits for the computation of that label. The former solution is costly if d is large, e.g., it requires $\Omega(\log n)$ bits if $d = \Omega(\log n / \log\log n)$. The latter solution is costly if the path from u to w contains lots of nodes with large degree (such a path can contain up to $\Omega(\log n)$ nodes of degree $\Omega(\log n / \log\log n)$) because the address of w must then contain information about each of these nodes. Our strategy will mix these two solutions.

3.1 Address Assignment

Let $f(n) = \Theta(\log n / \log\log n)$ (the reason for this setting will appear clear in forthcoming proofs). For any node u, we define the *light-path* of u as the sequence $\text{lpath}(u)$ such that $\text{lpath}(u) = ()$ if $u = r$, and if v is u's parent, then

$$\text{lpath}(u) = \begin{cases} \text{lpath}(v) & \text{if } \text{rank}(v,u) \leqslant f(w(v)); \\ (\text{lpath}(v), \text{port}(v,u)) & \text{otherwise.} \end{cases}$$

Roughly speaking, $\text{lpath}(u)$ contains only the large port numbers encountered while going from r to u. Let $\text{cpath}(u) = (\text{rank}(u_1,v_1), \ldots, \text{rank}(u_k,v_k))$. We define $E(u)$ as the binary string of length k such that

$$E(u)[i] = \begin{cases} 0 \text{ if } \text{rank}(u_i,v_i) \leqslant f(w(u_i)); \\ 1 \text{ otherwise.} \end{cases}$$

The address of u is similar to the one for the designer-port model, though it requires two extra fields, that is

$$\ell(u) \;=\; \langle \text{id}(u), \text{cpath}(u), E(u), \text{lpath}(u) \rangle \;.$$

The local data structure of u is similar to the one for the designer-port model, though it requires an extra field $P(u)$, i.e.,

$$\text{table}(u) \;=\; \langle \text{id}(u), w(u), w_1(u), k, P(u) \rangle \;.$$

$P(u)$ consists of an array of $f(w(u))+1$ entries that are port numbers such that

$$P(u)[i] = \begin{cases} \text{port}(u,x) & \text{if } i \in [1, f(w(u))] \text{ where } x \text{ satisfies } \text{rank}(u,x) = i; \\ 0 & \text{if } i = 0 \text{ and } u = r; \\ \text{port}(u,v) & \text{if } i = 0 \text{ and } u \neq r, \text{ where } v \text{ is the parent of } u. \end{cases}$$

3.2 Routing Algorithm

Upon reception of a message of destination y, the routing decision at x is described by a function $\textsc{Route2}(x, \ell(y)) \in [0, \deg(x)]$ that returns the link label on which the message has to be forwarded from x. As under the designer-port model, we make use of the $(|\text{cpath}(x)+1|)$-th element p of $\text{cpath}(y)$ as follows. Let

$$q = \begin{cases} P(x)[p] & \text{if } E(y)[p] = 0; \\ j\text{-th element of } \text{lpath}(y) & \text{if } E(y)[p] = 1, \text{ where } j = \text{ones}_{E(y)}(p). \end{cases}$$

We then set

$$\textsc{Route2}(x, \ell(y)) = \begin{cases} 0 & \text{if } \text{id}(y) = \text{id}(x); \\ P(x)[0] & \text{if } \text{id}(y) < \text{id}(x) \text{ or } \text{id}(y) \geqslant \text{id}(x) + w(x); \\ P(x)[1] & \text{if } \text{id}(x) < \text{id}(y) \leqslant \text{id}(x) + w_1(x); \\ q & \text{otherwise.} \end{cases}$$

3.3 Correctness

Lemma 6. *For every pair of nodes x, y, the routing algorithm described by* Route2 *routes any message from x to y along a shortest path.*

Proof. The first three cases are trivial: x knows both the link leading to its parent ($P(x)[0]$) and the link leading to its heaviest child ($P(x)[1]$). Only the fourth case requires some attention. Similarly to Lemma 2, the $(|\text{cpath}(x)+1|)$-th element of $\text{cpath}(y)$ is $p = \text{rank}(x,z)$ where (x,z) is the edge leading from x toward y. There are two cases:
Case 1. $E(y)[p] = 0$. Then $i = \text{rank}(x,z) \leqslant f(w(x))$, and thus $P(x)[i] = \text{port}(x,z)$ by definition of P. The function Route2 returns the correct answer in this case.
Case 2. $E(y)[p] = 1$. Then $i > f(w(x))$ and $\text{port}(x,z)$ is contained in $\text{lpath}(y)$. Note that the j-th port number along the clean-path from r to y is stored in $\text{lpath}(y)$ if and only if $E(y)[j]$ is set to 1. The number of 1-entries in $E(y)$ up to, and including, position p indicates how many port numbers are stored in $\text{lpath}(y)$. This allows to extract the j-th one by computing $j = \text{ones}_{E(y)}(p)$. The function Route2 returns the correct answer in this case as well, completing the proof. ∎

3.4 Implementation

The next lemma shows that both $\ell(u)$ and table(u) can be encoded on slightly less bits than $\log^2 n$. Recall that $f(n) = \Theta(\log n / \log\log n)$.

Lemma 7. *For every node u, its address $\ell(u)$ and its local data structure* table(u) *are both of size $O(\log^2 n / \log\log n)$ bits.*

Proof. Recall that $\ell(u) = \langle \mathrm{id}(u), \mathrm{cpath}(u), E(u), \mathrm{lpath}(u)\rangle$. The length of $E(u)$ is bounded by $|\mathrm{cpath}(u)|$. Thus the first three fields are all bounded by $O(\log n)$ bits. Assume that $\mathrm{lpath}(u) = (\mathrm{port}(u_1, v_1), \ldots, \mathrm{port}(u_t, v_t))$. We code $\mathrm{lpath}(u)$ via an array of t entries, where each entry is on $\lceil \log n \rceil$ bits. We have: $\mathrm{port}(u_i, v_i) \leqslant w(u_i)$, $w(u_{i+1}) \leqslant w(v_i)$, and $\mathrm{rank}(u_i, v_i) > f(w(u_i))$. By Lemma 1, we also have $\mathrm{rank}(u_i, v_i) \leqslant w(u_i)/w(v_i)$. Hence, $w(u_{i+1}) < w(u_i)/f(w(u_i))$. Let $g(n) = n/f(n)$. We have $w(u_{i+1}) < g(w(u_i))$. Thus, if we denote by $g^{(i)}(n)$ the i-th iteration of g, i.e., $g^{(0)}(n) = n$, and $g^{(i+1)}(n) = g(g^{(i)}(n))$, then

$$w(u_t) \;<\; g(w(u_{t-1})) \;<\; \cdots \;<\; g^{(t-1)}(w(u_1)) \;\leqslant\; g^{(t-1)}(n)\ .$$

Since $w(u_t) \geqslant 1$, it follows that $g^{(t-1)}(n) \geqslant 1$. Since $f(n) = \Theta(\log n / \log\log n)$, we have $g(n) = \Theta(n \log\log n / \log n)$, and therefore one can easily check that $t = O(\log n / \log\log n)$. Thus $\mathrm{lpath}(u)$ is coded by $O(t \log n) = O(\log^2 n / \log\log n)$ bits.

Recall that table$(u) = \langle \mathrm{id}(u), w(u), w_1(u), |\mathrm{cpath}(u)|, P(u)\rangle$. The first four fields are on $O(\log n)$ bits each. $P(u)$ is composed of $f(w(u)) + 1$ entries, each one on $\lceil \log n \rceil$ bits. Hence $P(u)$ is of size at most $O(\log^2 n / \log\log n)$, completing the proof. ∎

Theorem 2. *In the fixed-port model, every n-node tree supports a direct shortest-path routing scheme with $O(\log^2 n / \log\log n)$ bits for both the addresses and the local memory spaces (thus a memory-free shortest-path routing scheme with $O(\log^2 n / \log\log n)$ bits addresses), and $O(1)$ routing time. Moreover, the pre-processing time is $O(n \log n)$.*

Proof. Given the implementation of Lemma 7, Route2 can be computed in constant time, using the functions ones and select. The pre-precessing time does not increase, even though the size of the labels is slightly larger. Indeed, the pre-processing time for the operations $\mathrm{ones}_S(\cdot)$ and $\mathrm{select}_S(\cdot)$ are still performed on strings S of length $O(\log n)$, and hence it costs $O(\log n)$ time per node for its setup (cf. Lemma 5). ∎

3.5 A Routing Scheme with Short Range Addresses

In this section, we focus on routing schemes using addresses of minimum size. In particular, we present a simple routing scheme with $[1, n]$-range addresses, and using $n + o(n)$ bits of local memory space. In Section 4 we will show that this scheme is optimal.

Theorem 3. *In the fixed-port model, every n-node tree supports a direct shortest-path routing scheme with $[1,n]$-range address, $n+o(n)$ bit memory space, and with constant routing time. The pre-processing time to setup this scheme is $O(n^2)$.*

Proof. Let T be an n-node tree such that, for every edge (u,v), $\mathrm{port}(u,v)$ has been fixed and cannot be modified. We consider T as rooted at an arbitrary node, say r. We define the address $\ell(u)$ of u by labeling the nodes with integers in $[1,n]$ using a DFS traversal of T performed as follows: $\ell(r)=1$, and, at each internal node u, for any two children v_1 and v_2 of u, if $\mathrm{port}(u,v_2) > \mathrm{port}(u,v_1)$, then v_1 is visited before v_2.

For any node u, we also define the value $m(u)$ as the largest value of $\ell(w)$ among all descendents w of u. We finally define $B(u)$ as the binary array of n bits such that $B(u)[i]=1$ if and only if u has a child v such that $\ell(v)=i$. The local data structure of a node u is $\mathrm{table}(u)=\langle \ell(u), m(u), B(u), p(u)\rangle$ where $p(u)=\mathrm{port}(u,v)$ with v the parent of u ($p(u)=0$ if $u=r$).

The routing algorithm ROUTE3 is the following:

$$\textsc{Route3}(x,\ell(y))=\begin{cases} 0 & \text{if } \ell(y)=\ell(x);\\ p(x) & \text{if } \ell(y)>m(x) \text{ or } \ell(y)<\ell(x);\\ q+b \text{ otherwise, where } q=\mathrm{ones}_{B(x)}(\ell(y)) \text{ and } b=\begin{cases} 0 \text{ if } q\geqslant p(x)\\ 1 \text{ otherwise.}\end{cases}\end{cases}$$

The correctness of the algorithm is trivial in the two first cases. For the third case, note first that if $v_1,\ldots,v_t$ are the children of x such that $\mathrm{port}(x,v_1) < \cdots < \mathrm{port}(x,v_t)$, and if y belongs to some T_{v_i}, then the message has to be transmitted to v_i and the corresponding port number is i if $x=r$ or if $i<p(u)$, and $i+1$ if $x\neq r$ or $i\geqslant p(u)$. The test "y belongs to T_{v_i}?" can be performed by testing whether $\ell(v_i)\leqslant \ell(y) < \ell(v_{i+1})$, that is by computing the number of 1-entries in $B(x)$ up to (and including) position $\ell(y)$. Thus ROUTE3 returns the correct answer in the third case too. (Note that if $x=r$, the condition $q\geqslant p(x)$ is always satisfied because $p(r)=0$ and $q\geqslant 0$, thus $b=0$.)

The addresses are in the range $[1,n]$, and the local data structure is of size $n+o(n)$, including the data structure for the computation of $\mathrm{ones}_{B(x)}(\cdot)$ in constant time. The routing time is therefore $O(1)$, and the pre-processing time is $O(n)$ per node (the time to compute $B(x)$ and required to compute the $o(n)$ bits auxiliary bits for the ones operation, cf. Lemma 5). ∎

4 Lower Bounds for Short Range Addresses

Computing lower bounds often requires the use of *routing strategy*. A routing strategy $\mathcal{R}$ is simply a function that returns, for every graph G of a given family $\mathcal{F}$, a routing scheme $R=\mathcal{R}(G)$ for G. We denote by $\textsc{Mem}(R,x)$ the set of all the data structures used by R in node x of G.

Let $\mathcal{M}$ be any set, and let k be an integer. A *k-protocol* w.r.t. $\mathcal{R}$ and $\mathcal{M}$ is a pair of functions (s, m) such that, for any $G \in \mathcal{F}$, and for any $x \in V(G)$, we have

$$(1)\ \ s(\mathcal{R}(G), G, x) \in \{0,1\}^k \ \text{ and } (2)\ \ m(s(\mathcal{R}(G), G, x), \textsc{Mem}(\mathcal{R}(G), x)) \in \mathcal{M}.$$

A k-protocol (s, m) is said *complete* if for any $D \in \mathcal{M}$, there exists $G \in \mathcal{F}$, and $x \in V(G)$ such that

$$m(s(\mathcal{R}(G), G, x), \textsc{Mem}(\mathcal{R}(G), x)) = D.$$

We can prove the following:

Lemma 8. *Given a set $\mathcal{M}$, and a routing strategy $\mathcal{R}$ on $\mathcal{F}$, if there exists a complete k-protocol, then there exists $G \in \mathcal{F}$ and $x \in V(G)$ such that the size of $\textsc{Mem}(\mathcal{R}(G), x)$ is at least $\log|\mathcal{M}| - k$ bits.*

Proof. Let (s, m) be a complete k-protocol. Each element $D \in \mathcal{M}$ can be coded as a pair $\langle S, M\rangle$, where $S = s(\mathcal{R}(G), G, x)$, $M = \textsc{Mem}(\mathcal{R}(G), x)$, $G \in \mathcal{F}$ and $x \in V(G)$ are such that $m(S, M) = D$. By construction, given its code $\langle S, M\rangle$, we can obtain D by applying the function m. Let L be the size of $\textsc{Mem}(\mathcal{R}(G), x)$. The size of the coding of $D \in \mathcal{M}$ described above is at most $k + L$. If $k + L < \log|\mathcal{M}|$, then two distinct elements of $\mathcal{M}$ have the same encoding $\langle S, M\rangle$: a contradiction. Therefore, $L \geqslant \log|\mathcal{M}| - k$. ∎

By comparison with the routing scheme in Theorem 3 (fixed-port model) and the routing scheme in [4] (designer-port model), the two following lower bounds are optimal.

Theorem 4.
In the fixed-port model, every shortest path routing strategy on n-node trees with $[1, n + o(n/\log n)]$-range addresses requires at least $n - o(n)$ bits for some node of some tree.
In the designer-port model, every shortest path routing strategy on n-node trees with $[1, n + o(\sqrt{n}/\log n)]$-range addresses requires at least $\alpha\sqrt{n} - o(\sqrt{n})$ bits for some node of some tree.

Proof. Let us denote by $\mathcal{T}_1$ the family of unlabeled n-node trees, and by $\mathcal{T}_2$ the family of n-node trees such that every out-going edge of every node is labeled by a unique integer between 1 and the degree of that node. Moreover, let $\mathcal{M}_1$ be the set of sequences of non-null integers $n_1, n_2, \ldots$ such that $\sum_{i\geqslant 1} n_i = n - 1$. Let $\mathcal{M}_2$ be the set of non-decreasing sequences of non-null integers $n_1, n_2, \ldots$ such that $\sum_{i\geqslant 1} n_i = n-1$. Note that $|\mathcal{M}_1| = 2^{n-1}$ and that $|\mathcal{M}_2| = 2^{\alpha\sqrt{n}+o(\sqrt{n})}$ with $\alpha = (\pi\sqrt{2/3})/\ln 2 \approx 3.7$. (The latter formula comes from the fact that the number of ways to write n in non-null summands is asymptotically $\frac{1}{4n\sqrt{3}}\, e^{\pi\sqrt{2n/3}}$, the well-known Harder-Ramanujan's formula, cf. [6, Equation (4.2.7) page 44].)

Proving the two lower bounds requiring very similar technique, we give their proofs simultaneously.

Let $\mathcal{T} \in \{\mathcal{T}_1, \mathcal{T}_2\}$. For an arbitrary shortest-path routing strategy $\mathcal{R}$ on $\mathcal{T}$ using $[1, n + r]$-range addresses (we take $r = o(\sqrt{n}/\log n)$ in the fixed-port model, and $r = o(n/\log n)$ in the designer-port model), we define a complete k-protocol (s, m) as follows. For every $T \in \mathcal{T}$ and every $x \in V(T)$, we define $s(\mathcal{R}(T), T, x) = S$ where S the set of all the addresses assigned by $\mathcal{R}(T)$ distinct from the address of x. We define $m(S, \textsc{Mem}(\mathcal{R}(T), x)) = (n_1, \ldots, n_d)$ where n_i is the number of nodes such that the routing function in x, applied to all the addresses in S, returns the port number i. (For the designer-port model the n_i's are sorted so that $n_1 \leqslant \cdots \leqslant n_d$).

By construction, the sequence returned by m is in $\mathcal{M}_1$ in the fixed-port model, and in $\mathcal{M}_2$ in the designer-port model. Moreover, the k-protocol (s, m) is complete because, for every sequence $(n_1, \ldots, n_d)$, there exists a tree $T \in \mathcal{T}$, and a node x of degree d in T such that, for every child u of x:

- In the fixed-port model: $\text{port}(x, u) = i$ implies $w(u) = n_i$;
- In the designer-port model: $w(u) = n_i$.

By Lemma 8, for any routing strategy $\mathcal{R}$ with $[1, n+r]$-range addresses, there is a tree $T_0 \in \mathcal{T}$ and a node $x_0 \in V(T_0)$, such that the local memory space of x_0 is at least $\log |\mathcal{M}| - k$. It remains to show that $k = o(n)$ in the fixed-port model, and $k = o(\sqrt{n})$ in the designer-port model.

There are at most $\binom{n+r}{n-1}$ possible sets of $n-1$ addresses in $[1, n+r]$. Thus $k \leqslant \log \left\lceil \binom{n+r}{n-1} \right\rceil$. Since $n + r \leqslant 2(n-1)$, we have

$$\binom{n+r}{n-1} = \binom{n+r}{r+1} \leqslant (2(n-1))^{r+1} .$$

Therefore, $k \leqslant r \log n + O(r + \log n)$, that is $k = o(n)$ if $r = o(n/\log n)$, and $k = o(\sqrt{n})$ if $r = o(\sqrt{n}/\log n)$, which completes the proof. ∎

5 Conclusion

We have shown that n-node trees support routing schemes with message headers, node addresses, and local memory space of size $O(\log n)$ bits, and such that every local routing decision is taken in constant time. Beside this result, we have shown that little variations of the address size (by constant multiplicative factors, or logarithmic additive factors) can have a tremendous impact on the local memory space. In other words, the multiplicative constants hidden by the big-O notations play a significant role on the address-range vs. memory-space trade-off for routing. We can actually show similar results for space vs. time trade-offs. E.g., for some trees, small headers (i.e., in $[1, n + o(\sqrt{n}/\log n)]$) and compact data structures, (i.e., of size $2 \log n$) yield situations for which routing along shortest-path is doable but requires exponential times. It is actually even possible to show that the problem of whether there exists a direct shortest-path routing scheme with $[1, n + o(\sqrt{n}/\log n)]$-range addresses, and using local data

structures of optimal size up to a multiplicative factor of $\sqrt{n}/\log n$ is undecidable for any enumerable family of graphs containing n-node trees. All these results will be presented in the full version of the paper. As a last concluding remark, we conjecture that Theorem 2 is optimal, i.e., $\Omega(\log^2 n/\log\log n)$ bits for the addresses and the local data structures is a lower bound under the fixed-port model.

Acknowledgment. The authors are thankful to Michel Balazard and Michel Mendès France for showing them reference [14].

References

1. B. Awerbuch and D. Peleg, *Sparse partitions*, in 31^{th} Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, 1990, pp. 503–513.
2. L. J. Cowen, *Compact routing with minimum stretch*, in 10^{th} Symposium on Discrete Algorithms (SODA), ACM-SIAM, 1999, pp. 255–260.
3. T. Eilam, C. Gavoille, and D. Peleg, *Compact routing schemes with low stretch factor*, in 17^{th} Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM PRESS, Aug. 1998, pp. 11–20.
4. C. Gavoille, *A survey on interval routing*, Theoretical Computer Science, 245 (2000), pp. 217–253.
5. C. Gavoille and S. Pérennès, *Memory requirement for routing in distributed networks*, in 15^{th} Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM PRESS, May 1996, pp. 125–133.
6. M. J. Hall, *Combinatorial Theory (second edition)*, Wiley-Interscience Publication, 1986.
7. J. I. Munro, *Tables*, in 16^{th} FST&TCS, vol. 1180 of Lectures Notes in Computer Science, Springer-Verlag, 1996, pp. 37–42.
8. J. I. Munro and V. Raman, *Succinct representation of balanced parentheses, static trees and planar graphs*, in 38^{th} Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, Oct. 1997, pp. 118–126.
9. D. Peleg, *Proximity-preserving labeling schemes and their applications*, in 25^{th} International Workshop, Graph - Theoretic Concepts in Computer Science (WG), vol. 1665 of Lecture Notes in Computer Science, Springer, June 1999, pp. 30–41.
10. D. Peleg and E. Upfal, *A trade-off between space and efficiency for routing tables*, Journal of the ACM, 36 (1989), pp. 510–530.
11. N. Santoro and R. Khatib, *Labelling and implicit routing in networks*, The Computer Journal, 28 (1985), pp. 5–8.
12. M. Thorup and U. Zwick, *Compact routing schemes*, in 13^{th} Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), ACM PRESS, July 2001. To appear.
13. J. van Leeuwen and R. B. Tan, *Interval routing*, The Computer Journal, 30 (1987), pp. 298–307.
14. R. Warlimont, *Factorisatio numerorum with constraints*, Journal of Number Theory, 45 (1993), pp. 186–199.

Online Packet Routing on Linear Arrays and Rings

Jessen T. Havill

Department of Mathematics and Computer Science
Denison University
Granville, OH 43023 USA
havill@denison.edu

Abstract. In contrast to classical offline k-k routing, the online packet routing problem allows for an arbitrary number of packets with arbitrary end points and release times. We study this problem on linear array and ring networks. We generalize an earlier result for the offline problem by showing that Farthest First (FF) scheduling is optimal with respect to makespan on linear arrays. We also show that two other algorithms (Longest in System (LIS) and Moving Priority (MP)) have competitive ratio 2 with respect to makespan on linear arrays. For bidirectional rings, we show that, the competitive ratio of shortest path routing combined with LIS or MP scheduling is in $[2.5, 3)$ and the competitive ratio of shortest path routing combined with FF scheduling is 2. The latter algorithm is optimal among deterministic memoryless algorithms and all algorithms of which we are aware in the literature.

1 Introduction

The problem of efficiently moving packets of data among network nodes has classically been studied in the context of k-k routing where it is assumed that all packets are known to an algorithm before any packets are sent, and each node sends and receives exactly k packets. The traditional goal of an algorithm is to construct a schedule for any instance with makespan (maximum completion time) at most equal to the optimal makespan for a worst case instance. In contrast, we are interested in a generalized online problem that differs from the classical problem in three respects. First, we allow an arbitrary number of packets in an instance. Second, we allow any number of packets to originate at or be delivered to any node. Third, we allow packet release times to be arbitrary. We are interested in *oblivious online* algorithms that assign routes at the source before future packets are known. Our algorithms are also *distributed* in the sense that packets are scheduled locally at each intermediate node without regard for packets that have not yet passed through that node.

We will evaluate our algorithms using the competitive ratio (see below) with respect to makespan. We choose makespan for a couple of reasons. First, we wish to compare algorithms for the online problem with known algorithms for

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 773–784, 2001.

the offline problem, using a similar objective function, to expose interesting similarities and differences between the two problems. Second, even though we allow for arbitrary release times, makespan is still an appropriate measure for many situations, especially if release times are close together. For instance, suppose we want to minimize the completion time of a job on a message passing parallel computer when that job is characterized by a burst of packets. Then minimizing the makespan of the packets may be more important to minimizing the job completion time than minimizing, say, the maximum or average flow time of the packets. Of course, maximum and average flow time are also important criteria to study. We will mention some preliminary observations on these criteria later in the paper and leave further work as an important area for future research.

1.1 Problem Definition

In this paper, we consider full-duplex linear array and ring interconnection networks (and, by extension, networks with in-degree one). A linear array network contains n nodes labeled $\{0, 1, \ldots, n-1\}$ and $m = 2(n-1)$ directed links $\{(i, i+1), (i+1, i) : i = 0, 1, \ldots, n-2\}$. A ring network has n nodes and $m = 2n$ directed links $\{(i, (i+1) \bmod n), ((i+1) \bmod n, i) : i = 0, 1, \ldots, n-1\}$. The input to the problem is a sequence of packets $\sigma = p_1, p_2, \ldots, p_k$, ordered by release time. Each $p_j = (s_j, t_j, a_j)$, where s_j is the packet's source node, t_j is the packet's destination node, and a_j is the packet's release time. Let P_j be the (monotonic) route assigned to packet p_j, and let $P_j(i)$ denote the i^{th} link in the route.

Each network node with incoming links has a queue for each of its outgoing links to temporarily store packets being forwarded over that link. During each discrete time step $t \geq 1$, which represents the continuous time interval $(t-1, t]$, a node must decide whether each packet in a link's queue should wait in the queue or be one of at most w packets forwarded over that link during the next time step. If a packet arrives at a node during time step t, it will cross the next link on its path no earlier than during time step $t+1$. A schedule for packet p_j is a function S_j where $S_j(i)$ is the time step during which packet p_j will cross link $P_j(i)$. We assume that no packet is delayed due to a full queue at its next link. Rather, packets are delayed only when there are already w packets assigned to the next link or the algorithm decides to hold the packet for another reason. The overall schedule must obey the capacity constraints of the links during each time step.

Let C_j denote the completion time of packet p_j. The goal of an algorithm A is to minimize the makespan of its schedule, $C_A(\sigma) = \max_j C_j$, given a request sequence σ. Let $C^*(\sigma)$ be the optimal makespan for the same request sequence. Then A is c-competitive if and only if, for all σ, $C_A(\sigma) \leq c \cdot C^*(\sigma) + a$, where a is a constant. The competitive ratio of A is defined to be $\sup_\sigma C_A(\sigma)/C^*(\sigma)$.

We observe some trivial lower bounds on the optimal makespan for an instance. First, let $\delta = \max_j\{a_j + |P_j^*|\}$, where P_j^* is the route assigned to packet p_j by an optimal algorithm, and let $\mu^* = \max_{e \in E} |\{j : e \in P_j^*\}|/w$ be the congestion of the set of routes assigned by an optimal algorithm. Then we know that the

makespan of any optimal schedule must be at least $\Delta = \max\{\lceil \mu^* \rceil, \delta\}$. Similarly, define μ to be the congestion of the set of routes assigned by a particular online algorithm. If each packet has only one route, then clearly $\delta = \max_j \{a_j + d_j\}$, where d_j is the distance between s_j and t_j, and $\Delta = \max\{\lceil \mu \rceil, \delta\}$.

1.2 Packet Scheduling Algorithms

We will study three packet scheduling algorithms that are all *greedy* in the sense that a packet is delayed at a link e during time step t only if w other packets are already traversing e during t. First, LIS (Longest In System) [2] is the online scheduling algorithm that schedules each packet as early as possible on each link on its path, in the order it appears in the sequence. In other words, a packet p_j is given priority over packets p_i with $i > j$. (We note that this algorithm requires a global clock.) The second algorithm, MP (Moving Priority), gives packets passing through a node priority over packets originating at the node. Unlike LIS, MP does not rely upon a global clock. MP is a relaxed version of LIS on networks with in-degree one since every schedule constructed by LIS could have also been constructed by MP. Lastly, we study the FF (Farthest First) algorithm, which gives priority to packets that have the farthest distance yet to travel to their destinations.

1.3 Past Research and Our Results

Mansour and Patt-Shamir [8] proved that when packets follow shortest paths, *any* greedy scheduling algorithm guarantees that each p_j will arrive at its destination within $d_j + \lfloor (k-1)/w \rfloor$ steps, even in online instances. Cidon, et al. [3] showed that greedy policies cannot achieve this bound on an arbitrary set of paths. LIS was previously studied by Mao and Simha [9] and Rivera-Vega, et al. [10] who showed that LIS achieves the bound in [8] on shortest paths when $w = 1$. Valiant and Brebner [12] showed that MP delays each packet at most $k-1$ times on a linear array. We show that, on networks with in-degree one, LIS ensures that any packet p_j arrives at its destination by time $a_j + |P_j| + \lceil \mu_j \rceil - 1$, where μ_j is the maximum number of packets with smaller indices that traverse a link on P_j, normalized by w. Similarly, we show that MP guarantees a completion time of $a_j + |P_j| + \lceil \mu \rceil - 1$. We use these results to show that each of LIS and MP has a tight competitive ratio of $2 - 1/\Delta$ on linear arrays. Kaufmann and Sibeyn [5] showed that FF optimally solves the k-k routing problem on a linear array by constructing a schedule with makespan $\max_{u<v}\{v - u + h(u, v)\} - 1$ for packets moving either left or right, where $h(u, v)$ is the number of packets passing through nodes u and v from a particular direction. We generalize this result for the online problem by showing that FF constructs an optimal schedule with makespan $\max_{v,t \le T_v} \{\lceil N_{\ge}(v, t)/w \rceil + t\} - 1$, where $N_{\ge}(v, t)$ is the number of packets that would arrive at node v at time t or later if they were not delayed and T_v is the latest time a packet would arrive at node v if it were not delayed.

For a related offline problem with deadlines, Lui and Zaks [6] showed a necessary and sufficient condition for an instance to have a schedule that meets all

deadlines on bottleneck-free networks with at most one path connecting any pair of nodes. Furthermore, they showed that if this condition holds, the algorithm that forwards packets with the closest deadlines finds that schedule. Adler, et al. [1] designed a distributed online algorithm that meets at least $O(1/\log k)$ of the deadlines met in an optimal schedule. For some special cases, their algorithm comes within a constant fraction of optimal.

The well-known worst case lower bound on makespan for the general k-k routing problem on a ring is $\max\{(n-1)/2, kn/4\}$. For the special case in which all k packets originating at a node have the same destination, Makedon and Symvonis [7] designed an algorithm that constructs a schedule with makespan at most $kn/4 + 5n/2$. In contrast, they showed that the greedy algorithm that assigns every packet to its shortest path requires at least $k\lfloor n/2 \rfloor$ steps in the worst case. Kaufmann and Sibeyn [4] designed an algorithm for the general k-k routing problem on a ring which schedules packets with FF and guarantees a makespan of $kn/4+\sqrt{n}$. Sibeyn [11] designed an algorithm for the online problem with worst case makespan at most $kn/3 + n/3$ for a k-k distribution. The key to these ring routing algorithms is to always send some packets on their long paths. However, for the online problem, we show that *none* of these algorithms can have a better competitive ratio than the algorithm that sends every packet on its shortest path and schedules the packets with FF. This algorithm has competitive ratio 2. More specifically, we show that the competitive ratio of an online algorithm is at least 2 if it always assigns a request to its shortest path if the shortest path length is bounded by some fraction of the ring size. Therefore, in order to do better, a deterministic online algorithm must make routing decisions adaptively based on packets seen so far. We also show that the competitive ratio of the online algorithm that uses shortest paths and schedules packets with LIS or MP is in $[2.5, 3 - 1/\Delta]$. For the special case in which all packets have the same source and release time, we show that this algorithm is 2 competitive.

2 Scheduling with LIS and MP

We begin by proving properties of LIS and MP on full-duplex linear array networks (and more generally, any network with in-degree one) and then use these results to prove bounds on the competitive ratio of algorithms that combine LIS or MP scheduling with shortest path routing on rings.

2.1 Linear Array Networks

We will prove the following theorem regarding LIS and MP on linear arrays:

Theorem 1. *The competitive ratio of each of* LIS *and* MP *is* $2 - \frac{1}{\Delta}$ *on networks with in-degree one.*

First, we give two lemmas, one for each algorithm, which state upper bounds on the completion time of any packet. We omit the proof of the second lemma to conserve space. To simplify notation, let $\mu_j = \max_{e \in P_j} |\{i : i \leq j, e \in P_i\}| / w$.

Lemma 1. LIS *guarantees that* $C_j \leq a_j + d_j + \lceil \mu_j \rceil - 1$, *for all* j.

Proof. For any j, $C_j = a_j + d_j + x$, $x \geq 0$. For contradiction, assume $x > \lceil \mu_j \rceil - 1$. First, consider $P_j(l)$, at the tail of which p_j is delayed last, and notice that

$$S_j(l) = a_j + l + x \,. \tag{1}$$

Also notice that our assumption implies that $l > 1$. If this were not so then, since LIS is greedy, w packets with indices less than j must cross $P_j(1)$ during each time step $a_j + 1, a_j + 2, \ldots, a_j + x$, implying that $\lceil \mu_j \rceil \geq \lceil (xw + 1)/w \rceil = x + 1$.

Now let $X(\tau) = \{p_i : i < j$ and p_i crosses $P_j(l)$ during time step $\tau\}$. Also, to simplify notation, let $t = S_j(l-1) + 1$. Then $X(t), X(t+1), \ldots, X(S_j(l) - 1)$ are the sets of packets (each with cardinality w) that delay p_j on link $P_j(l)$. Additionally, there may be sets $X(r), X(r+1), \ldots, X(t-1)$ with cardinality w. If these additional sets do not exist, then we let $r = t$. Formally, we define $r = \min\{\tau : 2 \leq \tau \leq t, |X(\tau)| = w$ for all $\tau, \tau + 1, \ldots, t\}$.

We now prove the following lemma under the assumption that $x > \lceil \mu_j \rceil - 1$.

Lemma 2. *If* $x > \lceil \mu_j \rceil - 1$ *then, for all* $p \in X(\tau)$, $r \leq \tau \leq S_j(l) - 1$, p *must have crossed link* $P_j(l-1)$ *previously.*

Proof. Suppose, for contradiction, that there is a packet $p_i \in X(\tau)$, for some $r \leq \tau \leq S_j(l) - 1$, that did not cross link $P_j(l-1)$. Then, $P_i(1) = P_j(l)$. Therefore, by the definition of LIS, if p_i was delayed, then w earlier packets must have crossed $P_j(l)$ during each time step $a_i + 1, \ldots, \tau - 1$. But since w packets also cross $P_j(l)$ during each time step $\tau, \ldots, S_j(l) - 1$ (if p_i was not delayed, then $\tau = a_i + 1$) and p_j crosses $P_j(l)$ during time step $S_j(l)$,

$$\begin{aligned}
\lceil \mu_j \rceil &\geq \left\lceil \frac{((S_j(l) - 1) - (a_i + 1) + 1)w + 1}{w} \right\rceil \\
&= (S_j(l) - 1) - (a_i + 1) + 2 \\
&= (a_j + l + x) - a_i && \text{by (1)} \\
&> x + 1 && \text{since } a_j \geq a_i \text{ and } l > 1 \,. \qquad \square
\end{aligned}$$

Continuing with the proof of Lemma 1, let $X = \bigcup_{r \leq \tau \leq t} X(\tau) \cup \{p_j\}$. By Lemma 2 and the fact that $l > 1$, the $(t - r + 1)w + 1$ packets in X crossed link $P_j(l-1)$ before time step t. Thus, at least one packet $p \in X$ must have crossed link $P_j(l-1)$ before time step $r - 1$. Since p crossed link $P_j(l)$ during a time step greater than $r - 1$, p must have been delayed at the tail of link $P_j(l)$ by w packets crossing link $P_j(l)$ during time step $r - 1$. This set of packets must be $X(r-1)$. But, by definition, $X(r-1)$ contains strictly less than w packets. $\square$

Lemma 3. MP *guarantees that* $C_j \leq a_j + d_j + \lceil \mu \rceil - 1$, *for all* j.

We now use Lemmas 1 and 3 to bound the competitive ratios from above.

Lemma 4. *The competitive ratio of each of* LIS *and* MP *is at most* $2 - \frac{1}{\Delta}$ *on networks with in-degree one.*

Proof. We consider only LIS. The proof for MP is very similar. Let p_j denote the last packet to complete in the schedule constructed by LIS. By Lemma 1, we know that $C_j \leq a_j + d_j + \lceil \mu_j \rceil - 1$. On the other hand, we know that the optimal makespan is at least Δ. Therefore, the competitive ratio of LIS is at most $(a_j + d_j + \lceil \mu_j \rceil - 1)/\Delta \leq (\delta + \lceil \mu \rceil - 1)/\Delta \leq 2 - 1/\Delta$. □

We conclude by bounding the competitive ratios from below.

Lemma 5. *The competitive ratio of each of* LIS *and* MP *is at least* $2 - \frac{1}{\Delta}$ *on networks with in-degree one.*

Proof. Let $w = 1$, $k = n - 1$, and $p_j = (0, j, 0)$, for all $1 \leq j \leq k$. Note that $\delta = n - 1$ and $\mu = \mu^* = n - 1$ since all packets must cross link $(0, 1)$. LIS and MP schedule the packets in this instance in order, so that the last packet arrives at node $n - 1$ at time $(n - 2) + (n - 1) = 2n - 3$. On the other hand, an optimal algorithm will schedule the packets in reverse order, achieving a makespan of $n - 1$. Thus, the competitive ratio of both algorithms is at least $(2n - 3)/(n - 1) = 2 - 1/(n - 1) = 2 - 1/\Delta$. □

2.2 Ring Networks

We define RING1 to be the ring algorithm that routes each packet on its shortest path and schedules packets with either LIS or MP. We prove that the competitive ratio of RING1 is between 2.5 and $3 - 1/\Delta$. For the special case where all packets originate at the same node, we can show that the algorithm is 2 competitive. (We will omit the proof of this theorem to conserve space.)

We first prove the general upper bound. Notice that every simple path on a full duplex ring is contained in one of two disjoint simplex rings. We will call the simplex ring with links pointing clockwise the *right ring* and the simplex ring with links pointing counter-clockwise the *left ring*. Let $A = \{j : P_j = P_j^*\}$ and $B = \{j : P_j \neq P_j^*\}$. Furthermore, partition A into A_l and A_r where A_l is the subset of indices in A of packets which the online algorithm routes in the left ring and A_r is the subset of indices in A of packets which the online algorithm routes in the right ring. Similarly, partition B into B_l and B_r. Therefore, RING1 routes the packets with indices in $A_l \cup B_l$ in the left ring and the optimal algorithm routes the packets with indices in $A_l \cup B_r$ in the left ring. A symmetric property holds for the right ring. For a set of requests S, let $\mu_S = \max_{e \in E} |\{p_j \in S : e \in P_j\}| / w$. μ_S^* is defined analogously for the optimal route assignment.

Lemma 6. $\mu_{B_l} \leq \mu_{B_l}^*$ *and* $\mu_{B_r} \leq \mu_{B_r}^*$.

Proof. We will show that $\mu_{B_r} \leq \mu_{B_r}^*$. The argument for B_l is symmetric. Let $(i, (i+1) \bmod n)$ denote a link that satisfies the definition of μ_{B_r}. Let $S \subseteq B_r$ be the subset of requests that are assigned to $(i, (i + 1) \bmod n)$ in the algorithm's route assignment. So $\mu_{B_r} = |S|/w$. Now partition the nodes of the ring into two sets: let $X = \{\iota : \iota = (i + \lceil n/2 \rceil + 1) \bmod n, (i + \lceil n/2 \rceil + 2) \bmod n, \ldots, i\}$ and let $Y = \{\iota : \iota = (i + 1) \bmod n, (i + 2) \bmod n, \ldots, (i + \lceil n/2 \rceil) \bmod n\}$. Notice that the source of every request in S is in X and the destination of every request in S is in Y, since the requests are following shortest paths in the right

ji	a_{ji}	$\alpha(j,i)$	$\frac{n}{2}-j$	l_{ji}	(s_{ji},t_{ji})
11	0	3	3	3	$(0,3)$
12	0	2	3	3	$(0,3)$
13	0	—	—	—	$(0,4)$
14	0	—	—	—	$(0,4)$
21	1	2	2	3	$(1,4)$
22	1	1	2	2	$(1,3)$
23	1	—	—	—	$(1,5)$
24	1	—	—	—	$(1,5)$

p_{ji}	a_{ji}	$\alpha(j,i)$	$\frac{n}{2}-j$	l_{ji}	(s_{ji},t_{ji})
31	2	1	1	4	$(2,6)$
32	2	1	1	3	$(2,5)$
33	2	—	—	—	$(2,6)$
34	2	—	—	—	$(2,6)$
41	3	1	0	4	$(3,7)$
42	3	1	0	4	$(3,7)$
43	3	—	—	—	$(3,7)$
44	3	—	—	—	$(3,7)$

Fig. 1. A small lower bound instance with $n = 8$.

ring. Therefore, in the optimal route assignment, since all the requests in S are sent in the opposite direction, every request in S must be assigned to the link $((i + \lceil n/2 \rceil + 1) \bmod n, (i + \lceil n/2 \rceil) \bmod n)$. Thus, $\mu^*_{B_r} \geq |S|/w = \mu_{B_r}$. □

Lemma 7. $\mu \leq 2\mu^*$.

Proof. The congestion incurred by RING1 is $\mu \leq \max\{\mu_{A_l} + \mu_{B_l},\ \mu_{A_r} + \mu_{B_r}\}$. Without loss of generality, suppose $\mu_{A_l} + \mu_{B_l} \geq \mu_{A_r} + \mu_{B_r}$. Then, by Lemma 6, $\mu \leq \mu_{A_l} + \mu_{B_l} \leq \mu_{A_l} + \mu^*_{B_l} \leq 2\mu^*$. □

Theorem 2. RING1 *is* $3 - \frac{1}{\Delta}$ *competitive on a full duplex ring.*

Proof. Let p_j denote the packet that arrives at its destination last in the online schedule. Since the schedule on the full duplex ring is equivalent to two disjoint schedules, each on a simplex ring, we know from Lemmas 1 and 3 that $C_j \leq a_j + |P_j| + \lceil \mu \rceil - 1 \leq \max_i \{a_i + |P^*_i|\} + \lceil \mu \rceil - 1 \leq \delta + 2\lceil \mu^* \rceil - 1 \leq 3\Delta - 1$. The second inequality follows because packets are assigned to their shortest routes by the online algorithm. The third inequality follows from Lemma 7 and the definition of δ. Thus, the competitive ratio of RING1 is at most $(3\Delta - 1)/\Delta = 3 - 1/\Delta$. □

Finally, we bound the competitive ratio from below by 2.5 as $n \to \infty$.

Theorem 3. *The competitive ratio of* RING1 *is at least* 2.5.

Proof. We define an instance consisting of $k = 2n$ packets p_{ji}, where $1 \leq j \leq \frac{n}{2}$ and $1 \leq i \leq 4$. First, let $\alpha(j,i) = \lfloor (n - (j+i) + 1)/3 \rfloor$ and let

$$l_{ji} = \begin{cases} \min\{j - i + 2, n/2\}, & \alpha(j,i) \geq n/2 - j \\ \alpha(j,i) + 1, & \text{otherwise} \end{cases}.$$

Then, for $i = 1, 2$, we define $p_{ji} = (j-1, j + l_{ji} - 1, j-1)$, and for $i = 3, 4$, we define $p_{ji} = (j-1, j + n/2, j-1)$. In general, RING1 constructs a schedule with makespan $(5/2)n - 1$ for this instance and there is always a better schedule with makespan n. For arbitrarily large values of n, the instance demonstrates a lower bound of 5/2 for RING1. For example, consider the instance in Fig. 1 with $n = 8$. The schedule created by RING1, displayed in Fig. 2, has makespan 19. However, a better schedule, also displayed in Fig. 2, has makespan 8. □

As stated above, we can show that the algorithm is sometimes 2 competitive.

Theorem 4. *If all packets have the same arrival time and source, then any greedy algorithm that uses shortest paths is* 2 *competitive on a full duplex ring.*

	(0,1)	(1,2)	(2,3)	(3,4)	(4,5)	(5,6)	(6,7)	(7,0)
1	p_{11}							
2	p_{12}	p_{11}						
3	p_{13}	p_{12}	p_{11}					
4	p_{14}	p_{13}	p_{12}	p_{41}				
5		p_{14}	p_{13}	p_{42}	p_{41}			
6		p_{21}	p_{14}	p_{13}	p_{42}	p_{41}		
7		p_{22}	p_{21}	p_{14}		p_{42}	p_{41}	
8		p_{23}	p_{22}	p_{21}			p_{42}	
9		p_{24}	p_{23}	p_{43}				
10			p_{24}	p_{23}	p_{43}			
11			p_{31}	p_{24}	p_{23}	p_{43}		
12			p_{32}	p_{31}	p_{24}		p_{43}	
13			p_{33}	p_{32}	p_{31}			
14			p_{34}	p_{33}	p_{32}	p_{31}		
15				p_{34}	p_{33}			
16				p_{44}	p_{34}	p_{33}		
17					p_{44}	p_{34}		
18						p_{44}		
19							p_{44}	

	(0,1)	(1,2)	(2,3)	(3,4)	(4,5)	(5,6)	(6,7)	(7,0)	(0,7)	(7,6)	(6,5)	(5,4)	(4,3)	(3,2)	(2,1)	(1,0)
1	p_{11}								p_{13}							
2	p_{12}	p_{21}							p_{14}	p_{13}						p_{23}
3		p_{22}	p_{31}						p_{23}	p_{14}	p_{13}				p_{33}	p_{24}
4		p_{11}	p_{32}	p_{41}					p_{24}	p_{23}	p_{14}	p_{13}		p_{43}	p_{34}	p_{33}
5		p_{12}	p_{21}	p_{42}	p_{41}				p_{33}	p_{24}	p_{23}	p_{14}		p_{44}	p_{43}	p_{34}
6			p_{22}	p_{31}	p_{42}	p_{41}			p_{34}	p_{33}	p_{24}				p_{44}	p_{43}
7			p_{11}	p_{32}	p_{31}	p_{42}	p_{41}		p_{43}	p_{34}						p_{44}
8			p_{12}	p_{21}	p_{32}	p_{31}	p_{42}		p_{44}							

Fig. 2. RING1's schedule for the instance in Fig. 1 (top) and a better schedule (bottom).

3 Scheduling with FF

In this section, we generalize for the online case a result of Kaufmann and Sibeyn [5] by showing that scheduling with FF is optimal on linear arrays. We also show that the algorithm which combines FF with shortest path routing on rings is 2 competitive and optimal among all deterministic online algorithms that always send a packet on its shortest path if it is shorter than some constant fraction of n. This class of algorithms includes all of those of which we are aware in the literature, including the online algorithm of Sibeyn [11].

3.1 Linear Array Networks

We will say that a packet p will *ideally* arrive at node v at time t if $t = a_j + |v - s_j|$. Let $N(S, v, t) = |\{p_j \in S : v \in P_j \text{ and } a_j + |v - s_j| = t\}|$, the number of packets in a set S that would ideally arrive at node v at time t. Also, let $N_{\geq}(S, v, t) = \sum_{\tau=t}^{\infty} N(S, v, \tau)$ and $T(S, v) = \max\{t : N(S, v, t) > 0)\}$. When S is clear from the context, we will let $N(v, t) = N(S, v, t)$, $N_{\geq}(v, t) = N_{\geq}(S, v, t)$,

and $T_v = T(S, v)$. We first improve our trivial lower bound for scheduling on a linear array. We then show that FF achieves this bound and is therefore optimal.

Theorem 5. *Any scheduling algorithm constructs a schedule with makespan at least* $\max_{v,\, t \le T_v} \{\lceil N_{\ge}(v,t)/w \rceil + t\} - 1$ *on a linear array.*

Proof. Consider an arbitrary node v. During each time unit in $\{1, 2, \ldots, T_v\}$, at most w packets will reach node v. For any time unit $t \le T_v$, if there are more than $(T_v - t + 1)w$ packets that would ideally reach v at time t or greater then they could not all have reached v by time T_v. Rather, $N_{\ge}(v,t) - (T_v - t + 1)w$ packets must reach v after time T_v. Since this is true for any v and any $t \le T_v$, the time at which the final packet reaches its destination must be at least

$$\max_v \left\{ T_v + \max_{t \le T_v} \left\lceil \frac{N_{\ge}(v,t) - (T_v - t + 1)w}{w} \right\rceil \right\} = \max_{v, t \le T_v} \left\{ \left\lceil \frac{N_{\ge}(v,t)}{w} \right\rceil + t \right\} - 1.$$

□

Theorem 6. *The makespan of a schedule constructed by* FF *on a linear array is equal to* $\max_{v,\, t \le T_v} \{\lceil N_{\ge}(v,t)/w \rceil + t\} - 1$.

Proof. By the previous theorem, the lower bound holds. To prove the upper bound, consider an arbitrary node v and let $t_{\max}$ satisfy

$$\left\lceil \frac{N_{\ge}(v, t_{\max})}{w} \right\rceil + t_{\max} - 1 = \max_{t \le T_v} \left\{ \left\lceil \frac{N_{\ge}(v,t)}{w} \right\rceil + t \right\} - 1 \,. \tag{2}$$

The following inequalities follow from this definition:

$$\sum_{\tau=t}^{t_{\max}-1} N(v,\tau) \le (t_{\max} - t)w, \text{ for all } t \in \{1, 2, \ldots, t_{\max} - 1\} \tag{3}$$

$$\sum_{\tau=t_{\max}}^{t} N(v,\tau) \ge (t - t_{\max} + 1)w, \text{ for all } t \in \{t_{\max}, t_{\max} + 1, \ldots, T_v\} \tag{4}$$

If (3) were not true, then a smaller value of $t_{\max}$ would satisfy (2). If (4) were not true, then a larger value of $t_{\max}$ would satisfy (2).

We first show that, by the definition of $t_{\max}$ and (3), the packets counted in $\sum_{\tau=1}^{t_{\max}-1} N(v,\tau)$ will arrive at node v by time unit $t_{\max} - 1$. Notice that we can safely ignore any packets not counted in $N(v,t)$, for any t, since packets that will pass through node v will have priority over those that do not. If at most w packets would ideally arrive at node v at some time t, then they will all arrive at node v at their ideal time since no other packets will delay them. Now consider a group of $p > w$ packets that all want to arrive at node v at some time $t \in \{1, 2, \ldots, t_{\max} - 1\}$. By (3), for each such group, there must be at least $p - w$ additional available slots between $t + 1$ and $t_{\max} - 1$, inclusive, at which no packets would ideally arrive. Therefore, each of these p packets can arrive at node v at a unique time step at most $t_{\max} - 1$.

Next we show that the remaining packets, those that would ideally arrive at v after time $t_{\max}-1$, will all arrive at v by time $t_{\max}-1+\lceil N_{\geq}(v,t_{\max})/w\rceil$. Notice that, by (4), we can adjust the release time of each of these packets so that it will arrive at v no earlier than in the original schedule, and at a unique time step in $\{t_{\max}, t_{\max}+1, \ldots, t_{\max}-1+\lceil N_{\geq}(v,t_{\max})/w\rceil\}$. Note that the packets cannot finish any earlier with this modification in any schedule. Therefore, all packets that travel through v arrive there by time step $\lceil N_{\geq}(v,t_{\max})/w\rceil + t_{\max} - 1$ and the makespan of the schedule is at most $\max_{v,t\leq T_v}\{\lceil N_{\geq}(v,t)/w\rceil + t\} - 1$. □

This result reduces to that of Kaufmann and Sibeyn [5] if all a_j=0 and w=1:

$$\max_{v,t\leq T_v}\{N_{\geq}(v,t)+t\}-1 = \max_{v,t\leq v}\{h(v-t,v)+t\}-1 = \max_{u<v}\{h(u,v)+(v-u)\}-1\,.$$

3.2 Ring Networks

Let Ring2 be the ring algorithm that sends each packet on its shortest path and schedules packets with FF. We show that Ring2 has a tight competitive ratio of 2 and that this is better than any algorithm that routes a packet on its shortest path if it is shorter than a constant fraction of the ring size.

Theorem 7. *The competitive ratio of* Ring2 *is at most 2 on a ring.*

Proof. Let S_l denote the set of packets routed in the left ring by Ring2. Without loss of generality, suppose a packet in S_l has the latest completion time. Then $C_{FF} = \max_{v,t\leq T_v}\{\lceil N_{\geq}(S_l,v,t)/w\rceil + t\} - 1$. Let $v_{\max}, t_{\max} \leq T_{v_{\max}}$ be values that satisfy C_{FF}. We consider two cases (henceforth omitting S_l from notation):
Case 1: $\lceil N_{\geq}(v_{\max},t_{\max})/w\rceil \leq t_{\max}+1$.
In this case, $C_{FF} = \lceil N_{\geq}(v_{\max},t_{\max})/w\rceil + t_{\max} - 1 \leq 2t_{\max} \leq 2T_{v_{\max}} \leq 2C^*$.
Case 2: $\lceil N(v_{\max},t_{\max})/w\rceil > t_{\max}+1$.
For contradiction, suppose there exists an instance such that $C_{FF}/C^* > 2$.
Case 2a: If the optimal schedule directs $X \leq N(v_{\max},t_{\max}) - 1$ of the packets that are counted in $N(v_{\max},t_{\max})$ to the right then we know that $C^* \geq \lceil (N(v_{\max},t_{\max})-X)/w\rceil + t_{\max} - 1$. Since $C^* < (\lceil N(v_{\max},t_{\max})/w\rceil + t_{\max}-1)/2$ by assumption, this means that $X > (\lceil N(v_{\max},t_{\max})/w\rceil + t_{\max} - 1)/2$. (Notice that this is valid since, by assumption, $(\lceil N(v_{\max},t_{\max})/w\rceil + t_{\max} - 1)/2 < \lceil N(v_{\max},t_{\max})/w\rceil - 1$.) But, since all of the packets in X must traverse the edge $((v+\lceil n/2\rceil) \bmod n, (v+\lceil n/2\rceil - 1) \bmod n)$ in the right ring in the optimal schedule, it must be the case that $C^* \geq X > (\lceil N(v_{\max},t_{\max})/w\rceil + t_{\max} - 1)/2$ in the right ring. But this implies that $C_{FF}/C^* < 2$, a contradiction.
Case 2b: If the optimal schedule directs all of the packets that are counted in $N(v_{\max},t_{\max})$ to the right then we know that $C^* \geq \lceil N(v_{\max},t_{\max})/w\rceil$ in the right ring in the optimal schedule. This also implies a contradiction:

$$\frac{C_{FF}}{C^*} \leq \frac{\lceil N(v_{\max},t_{\max})/w\rceil + t_{\max} - 1}{\lceil N(v_{\max},t_{\max})/w\rceil} < 1 + \frac{t_{\max}-1}{t_{\max}+1} < 2\,. \qquad \square$$

Theorem 8. *The competitive ratio of any routing algorithm is at least* $2-\epsilon$*, for arbitrarily small positive* ϵ*, if it always assigns a request to its shortest path if the shortest path has length at most* βn*, for any* $\beta \in (0, 1/2]$*.*

Proof. Let $w = 1$. Consider a sequence of $k \geq (1-2\beta)n$ packets $(0, \lfloor \beta n \rfloor, 0)$. The algorithm will assign all these requests to their shortest path, resulting in a schedule with makespan at least $\lfloor \beta n \rfloor + k - 1$. On the other hand, a better schedule assigns $\lceil \alpha k \rceil$ packets to the shortest path and $\lfloor (1-\alpha)k \rfloor$ packets to the long path, where $\alpha = 1/2 + (1-2\beta)n/(2k)$. By Lemma 1, a LIS schedule on these routes will have makespan $\max\{\lfloor \beta n \rfloor + \lceil \alpha k \rceil - 1, (n - \lfloor \beta n \rfloor) + \lfloor (1-\alpha)k \rfloor - 1\} \leq (n+k)/2$. Therefore, the makespan of an optimal schedule will be at most $(n+k)/2$, and the competitive ratio for this instance is at least $2\left((\lfloor \beta n \rfloor + k - 1)/(n+k)\right)$, which approaches 2 for an arbitrarily large value of k. □

Theorem 9. *The competitive ratio of* RING2 *is exactly 2 on a ring and* RING2 *is optimal among algorithms which always assign a packet to its shortest path if the shortest path has length at most* βn*, for any* $\beta \in (0, 1/2]$*.*

3.3 Other Objective Functions

The consideration of other objective functions such as maximum flow time $(\max_j(C_j - a_j))$ and total flow time $(\sum_j (C_j - a_j))$ is an important direction for future research. Here we observe that FF is not optimal for these objective functions on linear arrays.

Observation 10. FF *is not optimal with respect to total flow time (or total completion time) on a linear array.*

Proof. Consider the following instance with $n = 3$ and $w = 1$: $p_0 = (0, 1, 0)$, $p_1 = (0, 2, 0)$, and $p_3 = (1, 2, 1)$. FF will assign p_1 to $(0, 1)$ during step 1, p_1 and p_2 to $(1, 2)$ during steps 2 and 3, and p_0 to $(0, 1)$ during step 2, giving a total flow time of 6 and total completion time of 7. On the other hand, LIS will assign p_0 to $(0, 1)$ during step 1, p_1 to $(0, 1)$ and $(1, 2)$ during steps 2 and 3, and p_2 to $(1, 2)$ during step 2, giving a total flow time 5 and total completion time 6. □

Observation 11. *The competitive ratio of* FF *is arbitrarily poor with respect to maximum flow time on a linear array.*

Proof. Consider the following instance on a 2 node linear array with $w = 1$: $p_0 = (0, 1, 0)$ and $p_j = (0, 1, j-1)$ for $j = \{1, 2, \ldots, k-1\}$. FF may schedule the packets in the following order: $p_1, p_2, \ldots, p_k, p_0$. The flow time of p_0 is k in this case, while the optimal maximum flow time is 2. □

4 Conclusions

We have studied how to route and schedule online packet instances on linear arrays and rings. We bounded the delay of two simple scheduling algorithms (LIS and MP) and proved that each has a competitive ratio of $2 - 1/\Delta$ on linear arrays. We also generalized for the online case a previous result by showing that

FF is optimal on linear arrays. We used these results to analyze the corresponding online algorithms that route packets on their shortest paths on rings. We showed that the ring algorithm that schedules with LIS or MP has a competitive ratio between 2.5 and $3 - 1/\Delta$, and the ring algorithm that schedules with FF has a tight competitive ratio of 2. The latter is optimal in the class of "memoryless" algorithms that always send a packet on its shortest path if its length is less than some constant fraction of the ring size. This class of algorithms includes all of which we are aware in the literature. In order to do better, a deterministic online algorithm would need to examine the network state and history, and assign a route and schedule accordingly. The investigation of such algorithms is an interesting area for future research. The study of other objective functions is also an important research direction. Since FF is not optimal with respect to maximum or total flow time, a study of other algorithms is required for these objectives. In addition, it would be interesting to extend this analysis to routing and scheduling on meshes and tori, and networks with arbitrary capacities.

References

[1] M. Adler, A. L. Rosenberg, R. K. Sitaraman, and W. Unger. Scheduling time-constrained communication in linear networks. In *Proc. ACM Symp. on Parallel Algorithms and Architectures*, pages 269–278, 1998.

[2] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queuing theory. In *Proc. ACM Symp. on Theory of Computing*, pages 376–385, 1996.

[3] I. Cidon, S. Kutten, Y. Mansour, and D. Peleg. Greedy packet scheduling. *SIAM Journal on Computing*, 24(1):148–157, 1995.

[4] M. Kaufmann and J. F. Sibeyn. Deterministic routing on circular arrays. In *Proc. IEEE Symp. on Parallel and Distributed Processing*, pages 376–383, 1992.

[5] M. Kaufmann and J. F. Sibeyn. Randomized multipacket routing an sorting on meshes. *Algorithmica*, 17:224–244, 1997.

[6] K.-S. Lui and S. Zaks. Scheduling in synchronous networks and the greedy algorithm. In *Proc. Int. Workshop on Distributed Algorithms*, pages 66–80, 1997.

[7] F. Makedon and A. Symvonis. Optimal algorithms for multipacket routing problems on rings. *Journal of Parallel and Distributed Computing*, 22(1):37–43, 1994.

[8] Y. Mansour and B. Patt-Shamir. Greedy packet scheduling on shortest paths. *Journal of Algorithms*, 14(3):449–465, 1993.

[9] W. Mao and R. Simha. Routing and scheduling file transfers in packet-switched networks. *Journal of Computing and Information*, 1(1):559–574, 1994.

[10] P. I. Rivera-Vega, R. Varadarajan, and S. B. Navathe. Scheduling data redistribution in distributed databases. In *Proc. IEEE Int. Conf. on Data Engineering*, pages 166–173, 1990.

[11] J. F. Sibeyn. Deterministic routing and sorting on rings. In *Proc. IEEE Int. Parallel Processing Symp.*, pages 406–410, 1994.

[12] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proc. ACM Symp. on Theory of Computing*, pages 263–277, 1981.

Faster Gossiping on Butterflies*

Jop F. Sibeyn

Computing Science Department, Umeå University, Sweden
http://www.cs.umu.se/~jopsi/

Abstract. Gossiping has been considered intensively for butterflies and "regular" butterflies (which have no wrap-around connections). In the telephone communication model, for a butterfly of order k, the best previous gossiping algorithms require $2^1/_2 \cdot k$ and $3 \cdot k$ communication rounds, respectively. By new asymptotic methods we break through these bounds, proving new bounds of $2^1/_4 \cdot k + o(k)$ and $2^1/_2 \cdot k + o(k)$.

1 Introduction

Gossiping. Collective communication operations frequently occur in parallel computing, and their performance often determines the overall running time of an application. One of the fundamental communication problems is *gossiping* (also called total exchange or all-to-all non-personalized communication). Gossiping is the problem in which every processing unit, PU, wants to send the same packet to every other PU. Said differently, initially each of the N PUs contains an amount of data of size h, and finally all PUs know the complete data set of size $h \cdot N$. Gossiping is used in all applications in which the PUs operate autonomously for a while, and then must exchange all gathered data to update their databases. Many aspects of the problem have been investigated for all kinds of interconnections networks [1,2,3,4,7,10,13].

Networks. We are considering butterflies and, lacking a better name, regular butterflies, illustrated in Figure 1. Each of them forms a parametrized class of networks. The k-th network is indicated by OBF_k (ordinary butterfly) and RBF_k (regular butterfly), respectively. OBF_k has $k \cdot 2^k$ nodes, all of degree 4; and RBF_k has $(k+1) \cdot 2^k$ nodes, which are of degree 2 and 4.

Ordinary butterflies are much considered networks because of their low-degree, regularity, small diameter and excellent routing properties. The RBF_k are less considered, but they are very important in the context of this paper, because the gossiping on OBF_k will be explained in terms of routing on RBF_k.

Communication Models. In the *telegraph* model a PU can be involved in only one communication operation: either receiving or sending, but not both. In the *telephone* model, a PU can communicate with only one of its neighbors at a time, but it can both send and receive during this communication. In this

* Most of the work on this paper has been performed during an appointment at and a visit to the Max-Planck-Institut für Informatik in Saarbrücken, Germany.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 785–796, 2001.

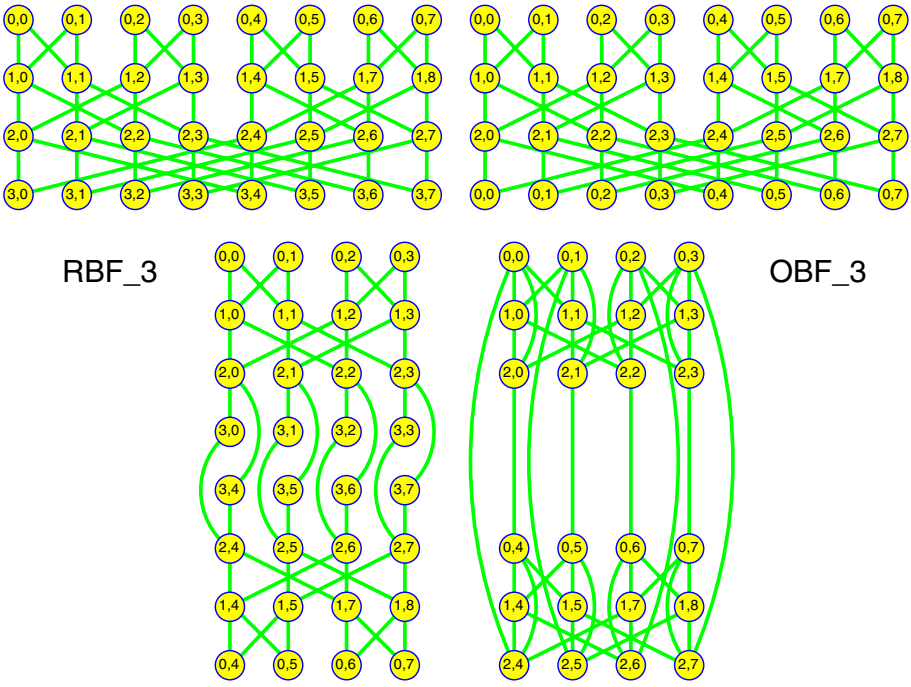


Fig. 1. RBF_3 and OBF_3. Two equivalent representations are given. In this paper, we will mostly refer to the lower one. In the upper representation of OBF_3, the top and bottom rows are to be identified.

paper we assume the telephone model, though it is easy to see that the same results can be obtained for the telegraph model as well (because they are based on the complexity of a certain broadcasting problem, which, of course, works under the assumptions of the telegraph model). As usual, we work under the *unit-cost* assumption, which means that communicating PUs can exchange an arbitrary amount of data in a single communication round.

Previous Work. The best theoretical results for gossiping in the telephone model on ordinary butterflies were given in [8]: OBF_k are given lower and upper bounds of $1.742 \cdot k$ and $2^{1}/_{2} \cdot k$, respectively. RBF_k is not considered, but clearly one needs at least $2 \cdot k$ communication rounds, because of the diameter of the network, and the algorithm for OBF_k can be modified to run in $3 \cdot k$ rounds. So, for the butterflies, there remains a considerable gap between the upper and lower bounds, and here one may hope to improve the leading constants. Experimental results are given in [12,5,6,11]. For $k \leq 12$ it is shown that most of the upper bounds in [8] are not sharp. However, though for de Bruijn and shuffle-exchange networks it appears that even the leading constant in the theoretical number of rounds can be improved, no such trend is discernible for butterflies.

New Results. The goal of this paper is to reduce the gap between upper and lower on the time consumption for gossiping on RBF_k and OBF_k as far as possible. As we have no ideas leading to improved lower bounds, we are trying to construct algorithms for gossiping on RBF_k and OBF_k with a time consumption of $c \cdot k + o(k)$ for the smallest possible constant c. We succeed in reducing c from 3 to $2^1/_2$ and from $2^1/_2$ to $2^1/_4$, respectively.

2 Gossiping on Column-Based Networks

The considered networks, can be thought to consist of a set of mutually disjoint *columns* (linear or circular arrays) of the same length covering all nodes, that are connected by the other connections. We refer to networks of this type as *column-based*. Because the columns all have the same size, the nodes can be divided in *rows*: subsets of the nodes consisting of one node from each column. Such a subdivision is not unique, but for a given subdivision in columns the structure of the network mostly allows only a single sensible one. The alternative representations for the ordinary and the regular butterfly networks shows that networks may be perceived in more than one way as column based. For the general idea these differences are not important. Throughout this paper k gives the number of nodes per column, that is the number of rows, numbered from 0 to $k-1$. For such a network we define an operation that is half-way between a broadcast and a gossip:

Definition 1 row-gossip(i,j) *is the operation of spreading the information initially stored in all PUs of row i to each single PUs of row j. A row-gossip is* row-restricted *if during each of the rounds operations are performed in $\mathcal{O}(1)$ consecutive rows only. Furthermore we require that the set of rows in which a row-restricted row-gossip is active progresses by one row after a constant number of rounds. The number of changes of running direction and running speed is bounded by a constant.*

The notions of *progress*, *running direction* and *running speed* are defined with respect to the fixed numbering of the rows. For example, the running speed is the reciprocal of the number of rounds before the maximum (minimum) of the indices of the rows in which operations are performed increases (decreases) by one.

The requirement of the number of turns and the evolution of the set of rows in which a row-gossip is active is no limitation: it does not make sense to let a row-gossip hang around in a limited set of rows. Once all connections have been used, no further gain of information can be made there. Using this definition, we can prove the following strong result which considerably simplifies the task of designing gossiping algorithms:

Theorem 1 *On a column-based network with k rows, if row-restricted row-gossip(i,j) can be performed in T rounds for all i and j, then gossiping can be performed in $T + o(k)$.*

Proof: In each column, x of the nodes are designated as bus stops. The value of x will be specified later. The bus stops are regularly interspaced with distance k/x in between. For example, one might take all nodes in the rows $l \cdot k/x$, for $0 \leq l < x$. $\sqrt{x}$ consecutive bus stops in the same column are grouped together in a zone. The bus stops are numbered with double indices (i_1, i_2), $0 \leq i_1, i_2 < \sqrt{x}$. i_1 giving the zone and i_2 the number within its zone. The algorithm consists of three phases:

1. Route all packets to all bus stops within their zones.
2. For all $0 \leq i_1, i_2 < \sqrt{x}$, perform *row-gossip*$((i_1, i_2), (i_2, i_1))$.
3. Spread the information from the bus stops to all PUs within their zones.

In a different context, a similar algorithm has been used in [9]. Phase 1 and phase 3 take at most $k/\sqrt{x}$ steps each: these operations can be performed by gossiping in the linear subarrays with $k/\sqrt{x}$ PUs each. Phase 2 in principle takes T rounds, as long as the most time consuming row-gossip, but we still must take care of possible conflicts between the operations of various row-gossips. When two row-gossips are active in overlapping rows, we give priority to the one moving towards the rows with higher indices. If two row-gossips running in the same direction are conflicting, then the slower one is delayed. Because our assumption that the number of changes of direction and speed is constant a row-gossip is delayed by any other row gossip for at most a constant number of rounds. The total number of row gossips equals x, so, the total incurred delay is $\mathcal{O}(x)$. Choosing $x = k^{2/3}$, the total time for the three rounds becomes $T + \mathcal{O}(k^{2/3})$. □

An important aspect here is that the gossiping time is only stated up to lower-order terms. In routing on meshes and the like this is a common way of specifying results, but in the theory on gossiping one mostly finds either precise results, like $\lfloor 5/2 \cdot k \rfloor + 1$, or very unspecific results like $\mathcal{O}(k)$. As in the area of gossiping on regular networks the main open questions are about the value of the leading constants, the later type of results is not appropriate, whereas the first type of results may be too hard to establish.

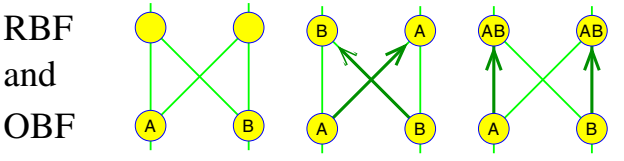


Fig. 2. Braiding on RBF and OBF. Initially, the sets of information A and B are each available in only one of the lower PUs. After the two routing operations these are available in both of the upper PUs.

We need some more vocabulary. By *walking* we mean the operation of forwarding information using only the edges of the columns. So, while walking

information spreads up- or downwards one row per round. The other operation is called *braiding*. It consists of two routing rounds, the first using connections *between* the columns, the second using connections *within* the columns. While braiding, the information progresses up- or downwards one row every two routing operations while spreading between the columns as well. This operation is illustrated in Figure 2. In the following $T_{\text{gos}}(RBF_k)$ denotes the number of rounds for gossiping on RBF_k. $T_{\text{gos}}(OBF_k)$ is defined analogously.

Lemma 1 $T_{gos}(RBF_k) \leq 3 \cdot k + o(k)$, $T_{gos}(OBF_k) \leq 2^{1}/_{2} \cdot k + o(k)$.

Proof: On RBF_k we proceed similarly, only here there are no "wrap-around" connections. Here the goals is to do at most k walking. We refer to the representation of RBF_k with $2 \cdot k + 2$ rows as in Figure 1 on the right. Without loss of generality we may assume that i lies in the upper half, that is, we assume $i \leq k$. We distinguish two main cases according to whether $i < j$ or not. If $i < j$ then we start by braiding for $2 \cdot k + \mathcal{O}(1)$ in the direction of increasing row indices. After this, the information walks to row j. If $j < i + k$, then the total time consumption is $3 \cdot k + i - k + \mathcal{O}(1)$, if $j > i + k$, then it is $k + j - i + \mathcal{O}(1)$. If $j < i$, then the initial information walks to row k, then braids until it reaches row 0 and then walks back to row j. This takes $3 \cdot k + j - i$ rounds in total. The proof for OBF_k is similar. □

3 Reduction of the Problem

The results in Lemma 1, which are not new, are not matching a lower bound. In [8], it is shown that gossiping on OBF_k asymptotically requires at least $1.7396 \cdot k$ rounds. For RBF_k, the diameter of $2 \cdot k$ gives a lower bound. In both cases there is a considerable gap with the results actually obtained.

Consider the gossiping schedules for OBF_k and RBF_k: the row-gossips on which they are based, are compositions of walking and braiding phases. Because braiding in a given row has the same effect as braiding in the corresponding row in the other half of the network, we will not distinguish these operations. Under this identification, if *row-gossip*(i, j) requires $T = 2 \cdot k + x$ rounds, we can say that it uses all k rows once for braiding, and in addition it uses x of them also for walking. This can be viewed as performing *row-gossip*$(0, 2 \cdot x + 1)$ on RBF_x. Refining this argument, we can prove

Lemma 2 *If row-gossip*$(0, 2 \cdot x + 1)$ *can be performed on* RBF_x *in* $(2+\delta) \cdot x + o(x)$ *rounds for all* x*, then* $T_{gos}(RBF_k) \leq (2 + \delta) \cdot k + o(k)$, $T_{gos}(OBF_k) \leq (2 + \delta/2) \cdot k + o(k)$.

Proof: In Theorem 1 we have shown that, except for lower-order terms, gossiping costs the same as the most expensive row-gossip. Let $T(x)$ denote the time for *row-gossip*$(0, 2 \cdot x + 1)$. For row-gossiping operations that are performed as a combination of walking through x rows and braiding through k rows, the cost can be written as $2 \cdot (k - x) + T(x)$. Here $2 \cdot (k - x)$ gives the time of braiding in

$k-x$ rows, while $T(x)$ gives the time for the remaining braiding and the walking. If $T(x)$ can be estimated on $(2+\delta)\cdot x + o(x)$, substitution gives that the time for row-gossip is bounded by

$$2\cdot(k-x) + (2+\delta)\cdot x + o(x) = 2\cdot k + \delta\cdot x.$$

In our above row-gossiping schedules the maximum value of x equals k and $k/2$ for RBF_k and OBF_k, respectively. Substitution gives the results. □

This lemma allows us to focus from the very general gossiping problem on one specific row-gossiping instance, which we will try to perform more efficiently in the following. To simplify even further, we will be working with a slightly different network in the following: RBF'_k is obtained by gluing two copies of RBF_k upside-down together. It has $2\cdot k+2$ rows and 2^k columns. RBF'_k is more regular, but clearly anything that can be done on RBF'_k can also be done on RBF_k. This regularity facilitates proving results like the following:

Lemma 3 *If row-gossip*$(0, 2\cdot k+1)$ *on* RBF'_k *can be performed in* t *rounds, then row-gossip*$(0, 2\cdot c\cdot k+1)$ *can be performed in* $c\cdot t$ *rounds on* $RBF'_{c\cdot k}$.

Proof: The proof goes by induction. The case $c=1$ is given. So, assume that *row-gossip*$(0, 2\cdot(c-1)\cdot k+1)$ can be performed in $(c-1)\cdot t$ rounds on $RBF'_{(c-1)\cdot k}$. Every column can be indexed by a pair (i,j), where i gives the k most important bits of the index and j the $(c-1)\cdot k$ least important bits. A PU is indicated by giving its row and column. Now we show that an information starting in an arbitrary PU $(0,(i,j))$ of row 0 reaches an arbitrary PU $(2\cdot c\cdot k+1,(i',j'))$ of row $2\cdot c\cdot k+1$. It is essential that the lowest and highest rows only work on the least important bits, and that the central rows only work on the most important bits. In the lowest $(c-1)\cdot k$ rows the information travels as it would do in $RBF'_{(c-1)\cdot k}$, when it would have to go from column j to column j'. It thus reaches some PU $((c-1)\cdot k,(i,j''))$. Then it performs the operations that it would perform in RBF'_k, when traveling from i to i'. Doing this, the information reaches some PU $((c+1)\cdot\cdot k+1,(i',j''))$. From there it continues to PU $(2\cdot c\cdot k+1,(i',j'))$ as it would do in the upper half of $RBF'_{(c-1)\cdot k}$. □

4 Coherent Row-Broadcasting

A row-broadcasting schedule for a column-based network is called *coherent*, if all operations that are performed during any given round in any given row use the "same" connections. In the case of RBF_k, this means that all packets with a given delay should use in a given round either straight or cross connections but not both. In general, coherent row broadcasting can only be defined if in every row the connections of the PUs can be allocated to classes so that all connections in a class can be used at the same time without causing conflicts.

Lemma 4 *If coherent row-broadcasting can be performed in* t *rounds, then also row-gossiping can be performed in* t *rounds.*

Proof: One row broadcasting scheme is started from each of the PUs in the row of origin. If in any given step more than one information must be forwarded from a given PU, then apparently these data have the same delay. Thus, by definition, they want to use the same connection, and can be combined. □

4.1 First Results

An example of a coherent row-broadcasting schedule is given in Figure 3. It spreads the information in RBF'_5 from the lower-left PU to all 32 PUs at the top in 13 rounds.

Lemma 5 $T_{gos}(RBF_k) \leq 2^3/_5 \cdot k + o(k)$, $T_{gos}(OBF_k) \leq 2^3/_{10} \cdot k + o(k)$.

Proof: According to Lemma 4, 32 of the coherent row-broadcasting schedules of Figure 3 can be combined to obtain *row-gossip*$(0, 2 \cdot k + 1)$ on RBF'_5. According to Lemma 3, c of these schedules can be combined to perform *row-gossip*$(0, 2 \cdot k + 1)$ on $RBF'_{5 \cdot c}$ in $13 \cdot c$ rounds. For k not a multiple of 5, we just add a few walking or braiding operations to complete the row-gossiping. Thus, for all k, *row-gossip*$(0, 2 \cdot k + 1)$ can be performed on RBF'_k in $13/5 \cdot k + \mathcal{O}(1)$ rounds. This schedule is so that at any given time packets from a given row-gossip are active in only three rows at a time. In that case, Theorem 1 and Lemma 2 give the result. □

Notice that this result is perfectly general, though we have only given a schedule for RBF'_5.

4.2 Towards $2\frac{1}{2} \cdot k$

We think that an important achievement of this paper is showing that asymptotically $T_{\text{gos}}(OBF_k) < 2^1/_2 \cdot k$. This was a long standing results, and one might have believed it to be tight. Nevertheless we also aim at establishing the smallest possible leading constants. So, we should come with bigger schedules, which at a certain point can no longer be found by puzzling and which cannot be verified by checking a picture.

A coherent row-broadcasting schedule allowing for d_1 delay in the lower half of the network and d_2 delay in the upper half can be represented by giving $(d_1 + 1) + (d_2 + 1)$ numbers of k bits each: the *wave vectors*. They indicate the column in which a packet starting in column 0 would end when it would exclusively use the by $0, \ldots, d_1$ rounds delayed wave for the lower half or the $0, \ldots, d_2$ delayed wave for the upper half. For example, the schedule in Figure 3 can be represented as $((0, 31, 0), (0, 21))$. This concise representation is an important additional advantage of working with coherent schedules. The wave vectors for the lower half are denoted w_i, $0 \leq i \leq d_1$, those for the upper half w'_i, $0 \leq i \leq d_2$. The regular structure of the network guarantees that if $((w_0, \ldots, w_{d_1}), (w'_0, \ldots, w'_{d_2}))$ is a schedule, than so is $((w'_0, \ldots, w'_{d_2}), (w_0, \ldots, w_{d_1}))$. It also guarantees that if there are schedules at all, then there is also a schedule with $w_0 = w'_0 = 0$.

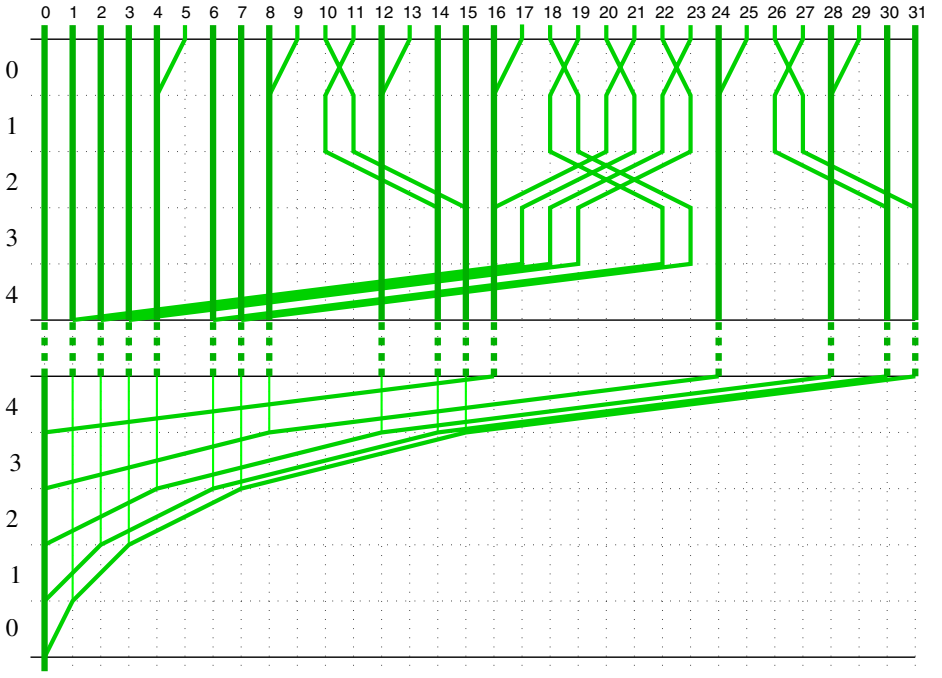


Fig. 3. Coherent row-broadcasting on RBF'_5 in 13 rounds. Indicated are the used connections. The thickest lines give the packets that are not delayed. The intermediate lines, give packets that are delayed once. The thinnest lines, only occurring in the lower half, indicate packets that are delayed twice. Thus, the lower half takes 7 and the upper half 6 rounds.

To obtain even better gossiping results, we have to investigate schedules for row-gossiping on RBF_k for $k > 5$. In order to speed up this process and to allow for exhaustive search, we have written the program described in Section 5. It shows that for $k = 4$ a coherent row-broadcast can reach at most 13 nodes with a schedule of 10 rounds. It also shows that for $k = 5$ there are many more schedules requiring 13 rounds (many of them related). For $k = 6$ there are no coherent schedules of length 15. Using such schedules at most 55 of the 64 columns can be reached. More generally, our program, after many improvements, has found schedules requiring $\lfloor 5 \cdot k/2 \rfloor + 1$ rounds, for all $k \leq 15$, and nothing better. For $k = 15$, one of the solutions is $((0, 32767, 0, 32767, 0), (0, 27483, 17846, 32461, 24))$. Just as in Lemma 5, this can be used to prove that gossiping on RBF_k can be performed in $38/15 \cdot k + o(k)$ rounds. $38/15 = 2.533$, so we already have come close to our main result:

Theorem 2 *If coherent row-broadcasting on RBF'_k can be performed in $\lfloor 5 \cdot k/2 \rfloor + 1$ rounds for all k, then $T_{gos}(RBF_k) \leq 2^1/_2 \cdot k + o(k)$, $T_{gos}(OBF_k) \leq 2^1/_4 \cdot k + o(k)$.*

Proof: If $T_{\rm gos}(RBF'_l) = \lfloor 5 \cdot l/2 \rfloor + 1$, for any constant l, the previous lemmas immediately give $T_{\rm gos}(RBF_k) = 5/2 \cdot k + c \cdot k/l$, for a small constant $c \geq 1$. So, for

obtaining the desired result, we should take l non-constant. We must be careful though, because this violates the assumption that the row-gossiping is active in only a constant number of rows. The best idea is to construct *row-gossip*$(0, 2 \cdot k + 2)$ from coherent row-broadcasting operations each spanning $k^{1/4}$ rows (see Lemma 3). This results in a schedule of length $2^{1}/_{2} \cdot k + \mathcal{O}(k^{3/4})$, which is active in at most $k^{1/4}$ rows at a time. If we choose the bus stops to lie $k^{1/2}$ apart with $k^{1/4}$ of them in each zone (see the proof of Theorem 1 for details), then the conflicts between the schedules result in at most another $\mathcal{O}(k^{3/4})$ delay. □

4.3 Completing the Argument

In the following, the schedule for coherent row-broadcasting on RBF'_k will be denoted $\mathcal{S}_k$. Its $(d_1+1)+(d_2+1)$ wave vectors are given in *binary*. $\mathcal{S}_k$ is written down as a matrix with a special format. For example,

$$\mathcal{S}_1 = \left(\begin{array}{c} 0 \\ \hline 0 \\ 1 \end{array}\right), \; \mathcal{S}_3 = \left(\begin{array}{c} 1\,0\,1 \\ \hline 0\,0\,0 \\ 0\,0\,0 \\ \hline 1\,1\,1 \end{array}\right), \; \mathcal{S}_5 = \left(\begin{array}{c} 1\,0\,1\,0\,1 \\ \hline 0\,0\,0\,0\,0 \\ \hline 0\,0\,0\,0\,0 \\ 1\,1\,1\,1\,1 \\ 0\,0\,0\,0\,0 \end{array}\right).$$

Here the lower rows give the movements in the lower half of RBF'_k and the upper rows the movements in the upper half. In each half the vectors have increasing delay away from the middle. There are two transitions to distinguish:

From $k = 4 \cdot l + 1$ to $k' = 4 \cdot l + 3$. The existing rows are extended on the right with twice the number they are ending on. So,

$$(\cdots\ 0) \longrightarrow (\cdots\ 0\,0\,0),$$
$$(\cdots\ 1) \longrightarrow (\cdots\ 1\,1\,1).$$

In addition, one new row is added to the upper half: $(1\ 0\ 1\ \cdots\ 0\ 1\ 0)$ for $k' = 8 \cdot m + 3$ and $(0\,1\,0\ \cdots\ 1\,0\,1)$ for $k' = 8 \cdot m + 7$. Applying these rules, we can derive $\mathcal{S}_7$ from $\mathcal{S}_5$:

$$\mathcal{S}_5 = \left(\begin{array}{c} 1\,0\,1\,0\,1 \\ \hline 0\,0\,0\,0\,0 \\ \hline 0\,0\,0\,0\,0 \\ 1\,1\,1\,1\,1 \\ 0\,0\,0\,0\,0 \end{array}\right) \longrightarrow \mathcal{S}_7 = \left(\begin{array}{c} 0\,1\,0\,1\,0\ \ 1\,0 \\ \\ 1\,0\,1\,0\,1\ \ 1\,1 \\ 0\,0\,0\,0\,0\ \ 0\,0 \\ \hline 0\,0\,0\,0\,0\ \ 0\,0 \\ 1\,1\,1\,1\,1\ \ 1\,1 \\ 0\,0\,0\,0\,0\ \ 0\,0 \end{array}\right).$$

From $k = 4 \cdot l + 3$ to $k' = 4 \cdot l + 5$. All existing rows are extended on the left with the same two numbers they so far were starting with:

$$(0\,0\ \cdots) \longrightarrow (0\,0\,0\,0\cdots),$$

$$(0\,1\ \cdots) \longrightarrow (0\,1\,0\,1\cdots),$$
$$(1\,0\ \cdots) \longrightarrow (1\,0\,1\,0\cdots),$$
$$(1\,1\ \cdots) \longrightarrow (1\,1\,1\,1\cdots).$$

In addition, one new row is added to the lower half: $(1\,1\ \cdots\ 1\,1)$ for $k' = 8 \cdot m + 1$ and $(0\,0\ \cdots\ 0\,0)$ for $k' = 8 \cdot m + 5$. Applying these rules, we can derive $\mathcal{S}_9$:

$$\mathcal{S}_7 = \left(\begin{array}{c} 0\,1\,0\,1\,0\,1\,0 \\ 1\,0\,1\,0\,1\,1\,1 \\ 0\,0\,0\,0\,0\,0\,0 \\ \hline 0\,0\,0\,0\,0\,0\,0 \\ 1\,1\,1\,1\,1\,1\,1 \\ 0\,0\,0\,0\,0\,0\,0 \end{array}\right) \longrightarrow \mathcal{S}_9 = \left(\begin{array}{cc} 0\,1 & 0\,1\,0\,1\,0\,1\,0 \\ 1\,0 & 1\,0\,1\,0\,1\,1\,1 \\ 0\,0 & 0\,0\,0\,0\,0\,0\,0 \\ \hline 0\,0 & 0\,0\,0\,0\,0\,0\,0 \\ 1\,1 & 1\,1\,1\,1\,1\,1\,1 \\ 0\,0 & 0\,0\,0\,0\,0\,0\,0 \\ & \\ 1\,1 & 1\,1\,1\,1\,1\,1\,1 \end{array}\right).$$

The computer has tested the $\mathcal{S}_k$ for all $k \leq 23$. For networks of this size (RBF'_{23} has $46 \cdot 2^{23}$ nodes) one could never find schedules by trying.

Lemma 6 *For all odd k, $\mathcal{S}_k$ gives a coherent row-broadcasting schedule for RBF'_k*

Proof: Clearly $\mathcal{S}_k$ is coherent. The proof that $\mathcal{S}_k$ is also a row-broadcasting schedule goes by induction. For $k \leq 5$, this has been checked. So, we may assume $\mathcal{S}_k$ and $\mathcal{S}_{k-2}$ are row-broadcasting schedules, when we are proving this for $\mathcal{S}_{k'}$, for an odd value $k' = k + 2 \geq 7$.

We consider the transition from $k = 4 \cdot l + 1$ to $k' = 4 \cdot l + 3$. In order to make the argument more concrete, we assume that $k' = 8 \cdot m + 7$ but, flipping zeroes and ones, the other case follows immediately. Let $d = l + 1$. Then we have delay vectors $w_0, \ldots, w_d$ and $w'_0, \ldots, w'_d$, each of length k'. We are mainly interested in $w_0, w_1, w'_{d-2}, w'_{d-1}$ and w'_d. Thus, essentially our situation looks like

w'_d		1 0
w'_{d-1}		1 1
w'_{d-2}		0 0
w_0		0 0
w_1		1 1

We will use that inside this block we find back $\mathcal{S}_k$. Starting with 0 0 (the least significant bits of w_0), we can continue with the operations of $\mathcal{S}_k$, and finish with either 1 1 (the least significant bits of w'_{d-1}) or 1 0 (the least significant bits of w'_d). This shows that all nodes with least significant bits 1 0 or 1 1 are reached. How about the nodes with least significant bits 0 0? These can be reached by starting with 0 0 (the least significant bits of w_0), and ending with 0 0 (the least

significant bits of w'_{d-2}), not using w'_{d-1} and w'_d, or by starting with 0 1 (bit 0 of w_0 and bit 1 of w_1), not using w_0, and ending with 1 0 (the least significant bits of w'_d). Neither of these two schedules reaches all nodes of RBF'_k, but, by splitting up further, it can be shown that the union of what is reached by each of them is the whole set. The nodes whose least significant bits are 0 1 can be reached by starting with 0 0 and ending with 0 1 (bit 1 of w'_{d-2} and bit 0 of w'_{d-1}), not using w'_{d-1} and w'_d, or by starting with 1 1, not using w_0, and ending with 1 0 (the least significant bits of w'_d). This is the same situation as before. The transition from $k = 4 \cdot l + 3$ to $k' = 4 \cdot l + 5$ is analyzed analogously. □

5 Program

We started with a program that was performing trivial exhaustive search within a specified range of values. This program, was useful for finding schedules up to $k = 7$. Because the problem grows extremely fast with k (the number of nodes to process for every solution grows exponentially, and the solution space grows even faster), only a considerably better approach can bring us further. We have introduced the following refinements, each of them allowing us to solve problems for k larger by two.

- Most promising first: for each of the wave vectors, all possible values where sorted according to the number of nodes that they allowed to reach, and then processed in this order.
- Testing only $((0, n-1, 0, \ldots), (0, *, *, \ldots))$: we discovered soon that it was no limitation to fix all wave vectors in the lower half and the first in the upper half. This gives a strong reduction of the search space.
- Specifying the number of different bits: by imposing that two consecutive wave vectors should be the same in not more than a specified number of bits, the search space was reduced even further.
- Setting thresholds: by imposing that after performing the wave vectors $w'_0, \ldots, w'_{d_2-1}$ in the upper half at least a certain (rather large) fraction of the nodes has to be reached, the search could be limited to a tree with degree less than 100.
- Bit parallelism: by using the bits of integers to represent sets and by fully exploiting bit-parallel operations, the memory consumption was minimized and the operations became much faster.

Without this program we would never have discovered the general structure presented in Section 4.3. The final version is available at `http://www.cs.umu.se/~jopsi/dprog/schedule.c`.

6 Conclusion and Further Research

We have shown how to solve gossiping on RBF_k in $5/2 \cdot k + o(k)$. This result is based on a reduction of gossiping to row-gossiping, which in turn was solved by overlapping shifted copies of a coherent row-broadcasting schedule. We see two further directions of research:

- It appears (but we are sure only for small k) that coherent row-broadcasting cannot be performed in less than $5/2 \cdot k$ rounds, but this does not exclude that row-gossiping might be performed faster.
- The algorithms are not practical because of the lower-order terms. However, we think that clever puzzling will reveal that all operations can be fit together so that no delays arise due to conflicts.

Acknowledgement. Michal Šoch has been reading and commenting parts of this paper. The comments of several ICALP referees were also helpful.

References

1. Bermond, J.-C., P. Fraigniaud, 'Broadcasting and Gossiping in de Bruijn Networks,' *SIAM Journal on Computing*, 23(1), pp. 212–225, 1994.
2. Delmas, O., S. Perennes, 'Circuit-Switched Gossiping in 3-Dimensional Torus Networks,' *Proc. 2nd Euro-Par Conference*, LNCS 1123, pp. 370–373, Springer-Verlag, 1996.
3. Fertin, G., 'Trade-Offs for Odd Gossiping,' *Proc. 6th Colloquium on Structural Information & Communication Complexity*, Proceedings in Informatics 5, pp. 137–151, Carleton Scientific, 1999.
4. Fraigniaud, P., J.G. Peters, 'Minimum Linear Gossip Graphs and Maximal Linear (Δ, k)-Gossip Graphs,' Techn. Rep. CMPT TR 94-06, Simon Fraser University, Burnaby, B.C., 1994. Available at `http://fas.sfu.ca/pub/cs/techreports/1994/`.
5. Fraigniaud, P., S. Vial, 'Approximation Algorithms for Broadcasting and Gossiping,' *Journal of Parallel and Distributed Computing*, 43(1), pp. 47–55, 1997.
6. Fraigniaud, P., S. Vial, 'Comparison of Heuristics for One-to-All and All-to-All Communication in Partial Meshes,' *Parallel Processing Letters*, 9(1), pp. 9–20, 1999.
7. Gvozdjak, P., J.G. Peters, 'Gossiping in Inclined LEO Satellite Networks,' *Proc. 6th Colloquium on Structural Information and Communication Complexity*, Proceedings in Informatics 5, pp. 166–180, Carleton Scientific, 1999.
8. Hromkovič, J., R. Klasing, B. Monien, R. Peine, 'Dissemination of Information in Interconnection Networks,' *Combinatorial Network Theory*, D.-Z. Du, D.F. Hsu (eds.), pp. 125–212, Kluwer Academic Publishers, 1996.
9. M. Kaufmann, R. Raman, J.F. Sibeyn, 'Randomized Routing on Meshes with Buses,' *Algorithmica*, 18, pp. 417–444, 1997.
10. Labahn, R., I. Warnke, 'Quick Gossiping by Telegraphs,' *Discrete Mathematics*, 126, pp. 421–424, North-Holland, 1994.
11. Beier, R., J.F. Sibeyn, 'A Powerful Heuristic for Telephone Gossiping,' *Proc. 7th Colloquium on Structural Information and Communication Complexity*, pp. 17–35, Proceedings in Informatics 7, Carleton Scientific, 2000.
12. Scheuermann, P., G. Wu, 'Heuristic Algorithms for Broadcasting in Point-to-Point Computer Networks,' *IEEE Transactions on Computers*, C-33(9), 1984.
13. Šoch, M., P. Tvrdík, 'Optimal Gossip in Store-and-Forward Noncombining 2-D Tori,' *Proc. 3rd International Euro-Par Conference*, LNCS 1300, pp. 234–241, Springer-Verlag, 1997.

Realizability and Verification of MSC Graphs

Rajeev Alur[1,2], Kousha Etessami[2], and Mihalis Yannakakis[2]

[1] Dept. of Computer and Info. Science, U. of Pennsylvania
[2] Bell Labs, Lucent Technologies
{alur,kousha,mihalis}@research.bell-labs.com

Abstract. Scenario-based specifications such as message sequence charts (MSC) offer an intuitive and visual way of describing design requirements. MSC-graphs allow convenient expression of multiple scenarios, and can be viewed as an early *model* of the system that can be subjected to a variety of analyses. Problems such as LTL model checking are known to be decidable for the class of *bounded* MSC-graphs.
Our first set of results concerns checking *realizability* of bounded MSC-graphs. An MSC-graph is realizable if there is a distributed implementation that generates precisely the behaviors in the graph. There are two notions of realizability, *weak* and *safe*, depending on whether or not we require the implementation to be deadlock-free. It is known that for a set of MSCs, weak realizability is coNP-complete while safe realizability has a polynomial-time solution. We establish that for bounded MSC-graphs, weak realizability is, surprisingly, undecidable, while safe is in EXPSPACE.
Our second set of results concerns verification of MSC-graphs. While checking properties of a graph G, besides verifying all the scenarios in the set $L(G)$ of MSCs specified by G, it is desirable to verify all the scenarios in the set $L^w(G)$—the *closure* of G, that contains the implied scenarios that any distributed implementation of G must include. For checking whether a given MSC M is a possible behavior, checking $M \in L(G)$ is NP-complete, but checking $M \in L^w(G)$ has a quadratic solution. For temporal logic specifications, considering the closure makes the verification problem harder: while checking LTL properties of $L(G)$ is PSPACE-complete and checking *local* properties has polynomial-time solutions, even for boolean combinations of local properties of $L^w(G)$, verifying acyclic graphs is coNP-complete and verifying bounded graphs is undecidable.

1 Introduction

Message Sequence Charts (MSCs) are a commonly used visual notation for describing message exchanges between concurrent processes. They have become popular among software engineers for early requirements specification. Recently MSCs have been standardized by ITU [12], and incorporated in modern software engineering notations such as UML [5]. In the simplest form, an MSC depicts the desired exchange of messages, and corresponds to a single (partial-order) execution of the system. In recent years, a variety of features have been introduced so that a designer can specify multiple scenarios conveniently. In particular,

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 797–808, 2001.

MSC-graphs allow MSCs to be combined using operations such as choice, concatenation, and repetition. MSC-graphs can be viewed as an early model of the system that can be subjected to formal analysis. This has motivated the development of algorithms for a variety of analyses including detecting race conditions and timing conflicts [2], pattern matching [9], detecting non-local choice [6], and model checking [3], and tools such as uBET [8] and MESA [4].

An MSC-graph consists of a graph G whose nodes are labeled by MSCs, and G is viewed as defining the set $L(G)$ of all MSCs obtained by concatenating the MSCs that appear along any (directed) finite path from the designated start node of G. It is worth noting that the traditional high-level model for concurrent systems has been communicating state machines. Both communicating state machines and MSC-graphs can be viewed as specifying sets of behaviors, but the two offer dual views; the former is a parallel composition of sequential machines, while the latter is a sequential composition of concurrent executions. The complexity of a variety of verification questions in the communicating-state-machines model has been well understood: typically the problems are undecidable, and we must assume a bound on the sizes of message-buffers to obtain decidability results. Recent results indicate that verification problems about MSC-graphs are also undecidable in general as a process can send potentially unbounded number of messages yet to be received [9,3]. The requirement for decidability, for problems such as LTL model checking, seems to be *boundedness*: in a bounded MSC-graph, in every cycle, for every pair of active processes p and q, there is a sequence of communications from p to q and back, ensuring that all the active processes stay roughly synchronized, thereby bounding the number of pending messages [3,4]. The boundedness property of an MSC-graph can be checked in time exponential in the number of processes [3]. In this paper, we study a variety of analysis problems for bounded MSC-graphs.

The first analysis question studied in this paper concerns a form of consistency, called *realizability*, of specifications given as an MSC-graph. As observed in [1], a set of MSCs can potentially imply other, distinct, MSCs whose communication pattern must be exhibited by any concurrent system that realizes the given MSCs. An MSC-graph G is said to be realizable if there exists a distributed implementation whose behaviors are precisely the ones specified by G. Unspecified, but implied, behaviors can be indicative of logical errors, and can be revealed by checking realizability. We prove that checking this form of realizability is, surprisingly, undecidable for bounded MSC-graphs by a reduction from the post correspondence problem. Intuitively, this is because, while a bounded graph ensures boundedness of buffers in the scenarios specified in the graph, it does not ensure boundedness of buffers in its distributed implementation where different processes can follow different paths in the graph.

We study a second form of realizability, called *safe* realizability, where the distributed implementation must be *deadlock-free*. Safe realizability is a stronger notion of realizability, and corresponds to inferring partial global behaviors from local views of specified MSCs. For a finite set of MSCs, checking weak realizability is coNP-complete, while checking safe realizability has a polynomial-time solution [1]. For bounded MSC-graphs, we show that checking safe realizability, unlike the weaker version, is decidable. We establish an upper bound of Ex-

PSPACE. We show the problem is PSPACE-hard, but matching the lower and upper bounds remains an open problem.

For the purpose of verification of an MSC-graph G, due to the gap between an MSC-graph and its implementation, besides $L(G)$, we also consider $L^w(G)$, the *weak-closure* of G, containing all MSCs implied by MSCs in G, as a possible semantics. As we will see, a verification question can have different answers and different complexities depending upon this choice of semantics.

Our first verification problem concerns testing whether a given scenario M is a possible behavior of a given MSC-graph G. This is relevant in identifying if a new scenario is already present in the existing specification, and also for detecting bugs if M specifies an undesired scenario. We show that the problem of verifying whether $M \in L(G)$ is NP-complete in general, but can be solved in polynomial-time if the number of processes is bounded. We establish that testing whether M is in the closure of $L(G)$ can be solved in quadratic time. This shows that it is easier to determine whether an MSC exists in the closure than in the originally given set, and furthermore, the questions about the implementation of G can sometimes be verified without constructing it.

Finally, we consider the model checking problem, where the model is given by an MSC graph G. When the semantics of G is $L(G)$, and the specification is given by an automaton accepting linearizations corresponding to "bad" behaviors, the problem is undecidable in general and PSPACE-complete for bounded graphs [3]. If the specification is given by "local" properties that do not distinguish between different linearizations of the same MSC, model checking can be solved in polynomial-time [10]. In this paper, we show that under the closure-semantics, the model checking questions become harder: for an acyclic graph (or even a set of MSCs) the problem is coNP-complete, and for bounded graphs the problem is undecidable, even for simple linearization-invariant local specifications.

2 MSCs and MSC Graphs

We start by recalling the definition of message sequence charts. Our definition captures the essence of the ITU standard MSC'96, and is analogous to the definitions of labeled MSCs given in [2,3]. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of processes, and Σ be a message alphabet. We write $[n]$ for $\{1, \ldots, n\}$. We use the label *send(i,j,a)* to denote the event "process P_i sends the message a to process P_j." Similarly, *receive(i,j,a)* denotes the event "process P_j receives the message a from process P_i." Define the set $\Sigma^S = \{send(i,j,a) \mid i,j \in [n] \ \&\ a \in \Sigma\}$ of *send labels*, the set $\Sigma^R = \{receive(i,j,a) \mid i,j \in [n] \ \&\ a \in \Sigma\}$ of *receive labels*, and $\Sigma = \Sigma^S \cup \Sigma^R$ as the set of *event labels*. A *Σ-labeled MSC M* over processes $\mathcal{P}$ is given by:

1. a set E of events which is partitioned into a set S of "send" events and a set R of "receive" events;
2. a mapping $p : E \mapsto [n]$ that maps each event to a process on which it occurs;
3. a bijective mapping $f : S \mapsto R$ between send and receive events, matching each send with its corresponding receive;

4. a mapping $l : E \mapsto \Sigma$ which labels each event such that $l(S) \subseteq \Sigma^S$ and $l(R) \subseteq \Sigma^R$, and furthermore for consistency of labels, for all $s \in S$, if $l(s) = send(i, j, a)$ then $p(s) = i$ and $l(f(s)) = receive(i, j, a)$ and $p(f(s)) = j$;
5. for each $i \in [n]$, a total order $\leq_i$ on the events of process P_i, that is, on the elements of $p^{-1}(i)$, such that the transitive closure of the relation

$$\leq \doteq \cup_{i\in[n]} \leq_i \cup \{(s, f(s)) \mid s \in S\}$$

is a partial order on E.

We require our MSCs to satisfy an additional *non-degeneracy* condition: there is no reversal of the order in which two *identical* messages sent by some process P_i are received by another process P_j. Observe that the information in such MSCs can be captured by a word over Σ that corresponds to the sequence of labels of a linearization consistent with the partial order $\leq$. To be precise, a word w over Σ is *well-formed* if all receive events have matching sends; and is *complete* if all send events have matching receives. Then, a word w over the alphabet Σ is a linearization of an MSC iff it is well-formed and complete (see [1]).

A natural way to structure multiple scenarios is to employ graphs whose nodes are MSCs. Formally, an MSC-graph G consists of a set V of vertices, a binary relation $\rightarrow$ over V, an initial vertex v^I, a terminal vertex v^T, and a labeling function μ that maps each vertex v to an MSC. The paths that start at the initial vertex and end at the terminal vertex represent the finite executions of the system modeled by the MSC-graph. To formally associate a set of MSCs with the MSC-graph G, we first have to define a concatenation operation on MSCs. Concatenation $M \cdot M'$ corresponds to a natural process-by-process pasting of the two MSCs M and M' together (see [3] for a formal definition). Then, we can associate an MSC with each path by concatenating MSCs corresponding to individual vertices. The language $L(G)$ of the graph is then all MSCs of the form $\mu(v_0) \cdot \mu(v_1) \cdots \mu(v_n)$, where $v_0 v_1 \ldots v_n$ is an accepting path in G. Since MSCs are uniquely characterized by their linearizations, we will also use $L(G)$ to denote the set of linearizations of the MSCs in it.

In general, the set $L(G)$ is not regular. The problematic case is when there is a cycle in the graph such that some process sends a message at some vertex in the cycle, but does not receive any message at any vertex in the cycle. For example, consider the MSC-graph with a single node with a self-loop, where the MSC associated with the node consists of a single message edge. The class of bounded MSCs avoids this problem. Given an MSC-graph G and a subset U of its vertices, define the communication graph H_U of U as follows: the set of vertices of H_U is the set P of all the processes, and there is an arc from process p to process q if p sends a message to q in the MSC $\mu(v)$ for some $v \in U$. For a set U of vertices, we denote by P_U the set of processes that send or receive a message in the MSC of some vertex in U, and call them the active processes of the set U. We call an MSC-graph *bounded* if for every cycle ρ of G, the subgraph of the communication graph H_ρ induced by the set P_ρ of active processes of the cycle is strongly connected. In other words, communication graph H_ρ on all the processes consists of one nontrivial strongly connected component and isolated nodes corresponding to processes that are inactive throughout the cycle. In [3],

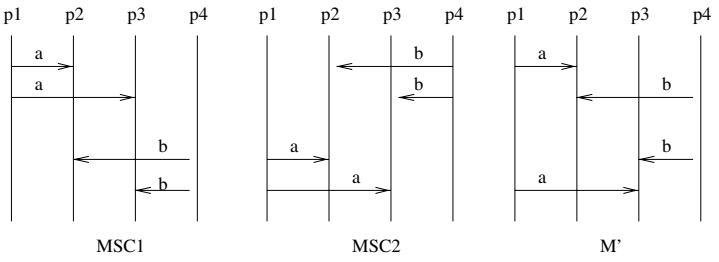


Fig. 1. Weak Inference

it is shown that if G is bounded, the set of linearizations of all the MSCs in $L(G)$ is regular, and can be generated by a nondeterministic automaton whose size is exponential in the size of G. The converse of the question, namely, characterizing regular languages using MSC graphs, is studied in [7].

3 Realizability

3.1 Weak Realizability

Consider the two MSCs MSC1 and MSC2 shown in Figure 1. Any distributed implementation that exhibits these two behaviors must also exhibit the behavior depicted by M'. This is because, as far as each process can locally tell, the scenario is proceeding according to one of the two given scenarios. Consequently, we say that the set of MSCs containing MSC1 and MSC2 (weakly) implies M' [1].

Formally, for an MSC M and a process P_i, let $M|_i$ denote the sequence of events belonging to the process P_i in M. Then, a set L of MSCs *weakly implies* an MSC M iff for all $i \in [n]$, there exists an MSC $M_i \in L$ such that $M|_i = M_i|_i$. The *weak closure* L^w of a set L of MSCs contains all the MSCs L weakly implies, and the set L is weakly realizable iff $L = L^w$. The notion of weak realizability was introduced in [1] in the context of the underlying communicating state machines that would realize the communication patterns specified by given MSCs. In particular, we had shown that L is weakly realizable iff there exist automata A_i, one per process P_i, asynchronously communicating with one another over FIFO buffers, such that the global product of these automata exhibits precisely the behaviors in L.

The notions defined above naturally extend to MSC-graphs. The MSC-graph G is said to be weakly realizable if the set $L(G)$ of MSCs is. Thus, a weakly realizable graph already contains all the implied scenarios. We show that checking weak realizability is undecidable for bounded graphs.

Theorem 1. *Given a bounded MSC graph G, checking if G is weakly realizable is undecidable.*

Proof. The proof is a reduction from the *Post Correspondence Problem* (PCP). The PCP is as follows: given a collection of pairs $\langle (v_1, w_1), (v_2, w_2), \ldots, (v_r, w_r) \rangle$, where $v_i, w_i \in \Sigma^*$, for some fixed finite alphabet Σ, with designated initial pair (v_1, w_1), determine whether there is a sequence of indices $i_2, \ldots, i_m$, such that

$$v_1 v_{i_2} \dots v_{i_m} = w_1 w_{i_2} \dots w_{i_m} \quad (1)$$

By examining the standard proof of undecidability for the PCP from the Turing machine halting problem, one can see that the constructed PCP instance has the property that if there is a solution then there is one where the one string is always a prefix of the other. In particular, the following version, call it OneSidedPCP, remains undecidable: determine whether there is a sequence of indices $i_2 \dots i_m$, such that equality 1 holds, and furthermore, for all $j \leq m$, the string $w_1 w_{i_2} \dots w_{i_j}$ is a prefix of the string $v_1 v_{i_2} \dots v_{i_j}$ (that is, the right string never overtakes the left one). We will reformulate OneSidedPCP slightly further to suit our purposes. Let *Relaxed PCP* (RPCP) be the following problem: given $\{(v_1, w_1), (v_2, w_2), \dots, (v_r, w_r)\}$, determine whether there are indices $i_1, \dots, i_m$ such that $x_{i_1} \dots x_{i_m} = y_{i_1} \dots y_{i_m}$, where $x_{i_j}, y_{i_j} \in \{v_{i_j}, w_{i_j}\}$, for some index i_l $x_{i_l} \neq y_{i_l}$, and for all $j \leq m$, $y_{i_1} \dots y_{i_j}$ is a prefix of $x_{i_1} \dots x_{i_j}$. The proof that RPCP is undecidable is omitted due to lack of space.

Now we reduce RPCP to weak realizability. Given a finite set L of MSCs, let L^* denote the MSC graph that consists of the complete graph with $|L|$ vertices one per MSC in the set L, dummy initial and terminal vertices v_I, v_T with empty MSC's, and edges from v_I to all vertices of L and from those to v_T. Thus, an MSC of this graph is simply a concatenation of MSC's from the set L. In the sequel, we say that a process p synchronously sends a message m to process q, if p sends m to q immediately followed by q sending the message m back to p. In figures, such messages will be depicted by double arrows.

Given an instance $\Delta = \{(v_1, w_1), \dots, (v_m, w_m)\}$ of RPCP, we build a set L of MSCs over 4 processes as follows. For a string u, let u^l denote the l'th character of the string. For each pair (v_i, w_i) we build two MSCs M_i^0 and M_i^1, which are depicted in Figure 2. Thus in M_i^0, process 1 sends synchronously $(i, 0)$ to process 2 then sends the index i to process 4, and then process 4 sends synchronously $(i, 0)$ to process 3. After that, process 2 synchronously sends the sequence of characters of v_i to process 3 (note we assume c is the length of v_i and d the length of w_i in the figure), M_i^1 is similar. Observe that the communication graph of each of these MSCs is strongly connected and involves all the processes, and hence, the MSC graph L^* is bounded.

Claim. $\Delta \in RPCP$ iff L^* is not weakly realizable.

□

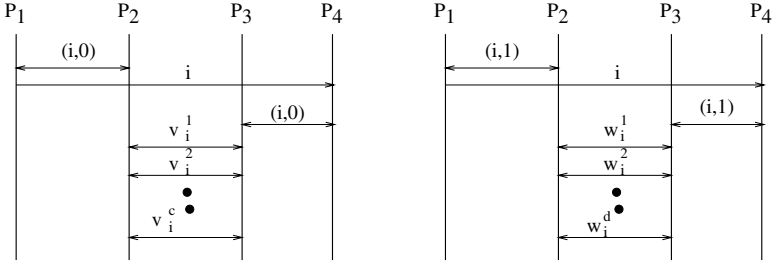


Fig. 2. MSCs M_i^0 and M_i^1

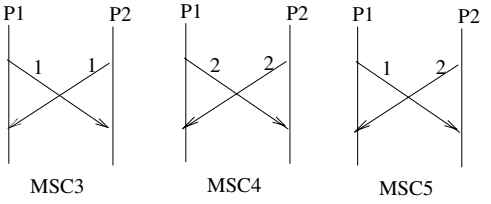


Fig. 3. Safe realizability

3.2 Safe Realizability

As a motivation for safe realizability, consider the MSCs in Figure 3. In MSC_3, both processes send each other the value 1, while in MSC_4, both processes send each other the value 2, and thus, they agree in both cases. From these two, we should be able to infer a partial scenario, depicted in MSC_5, in which the two processes start by sending each other conflicting values, and the scenario is then completed in some way. However, the set containing only MSC_3 and MSC_4 is weakly realizable. A closer examination reveals that the distributed implementation of these two scenarios can potentially deadlock when one process decides to send the message 1 while the other decides to send the message 2. We need a stronger version of implication closure.

Safe realizability detects such problems by considering *partial MSCs*, or equivalently, prefixes of linearizations. Let $pref(L)$ denote the set of prefixes of the MSCs or words in L. Then, a set L of MSCs *safely implies* a well-formed word w (i.e. a partial MSC) iff for all $i \in [n]$, there exists a word $v_i \in pref(L)$ such that $w|_i = v_i|_i$. The set L is safely realizable iff $pref(L)$ contains all the words safely implied by L. As shown in [1], L is safely realizable iff there exist automata A_i, one per process P_i, asynchronously communicating with one another over FIFO buffers, such that the global product of these automata is deadlock-free and exhibits precisely the behaviors in L.

The MSC-graph G is said to be safely realizable if the set $L(G)$ of MSCs is. For a finite set of MSCs, it is known that weak realizability is coNP-complete while safe realizability has polynomial-time solution. For bounded graphs, even though weak realizability is undecidable, checking safe realizability is decidable. In bounded graphs, if we consider the behaviors corresponding to the paths in the graphs, a process cannot be far ahead of its communication partner, thus keeping the buffer size bounded. While checking safe realizability, when we consider the possible interactions among local behaviors (i.e. projections) of different processes, if the communication buffer between any pair of processes exceeds this bound, we can immediately flag an error. In contrast, while checking weak realizability, even when the buffer size exceeds the bound, we need to check if there is a "complete" MSC that can extend this partial behavior. We establish an EXPSPACE upper bound, as well as PSPACE-hardness, for checking safe realizability, but its precise complexity class remains an open problem.

Theorem 2. *Checking safe realizability of a bounded MSC-graph is in* EXPSPACE.

Proof. We know that $L(G)$ is definable by an exponential sized automaton A each of whose states can be encoded in polynomial space. Likewise, we can build a concurrent product $A' = \Pi_i A_i$, where the A_i are the local automata formed by the projection of G onto process i, and then determinized and minimized. If $L(G)$ is safely realizable, then we know that A' is such a realization. Moreover, since G is bounded, there is a polynomial bound (actually, linear in the number of vertices of G) that we can place on the lengths of queues in A' such that if ever the queue length is exceeded we will know that the partial MSC which exceeded the bound is not a prefix of an MSC in $L(G)$. Thus, we first check to see whether there is an execution of A' in which the buffer bound is exceeded. This can be done in PSPACE by guessing a bad path. If there is such an execution, we halt and report that $L(G)$ is not realizable. Thus we assume that A' enforces the polynomial bound on the buffers. We can then "build" the complement automaton $\bar{A}$ for $L(G)$ (we don't actually build $\bar{A}$, but compute its states using the subset construction as we need them). We then need to know whether $L(A') \cap L(\bar{A})$ is empty or not. If it is, then A' realizes $L(G)$. If not, then $L(G)$ is not safely realizable. Since each state of $\bar{A}$ requires exponential size to encode, we can determine whether $L(A') \cap L(\bar{A})$ is nonempty in EXPSPACE by guessing an accepting path in each automaton. □

Theorem 3. *Checking safe realizability of a bounded MSC-graph is* PSPACE*-hard.*

Proof. We reduce the PSPACE-complete problem of determining whether a given NFA, A, accepts Σ^*, to checking safe realizability. Assume $A = \langle Q, \Sigma, \delta, q_0, l, F \rangle$ is Σ-labeled on states rather than on transitions, i.e., $l : Q \mapsto \Sigma$, and $\delta \subseteq Q \times Q$. Let $\Sigma = \{a_1, \ldots, a_k\}$ and $Q = \{q_1, \ldots, q_n\}$. We build from A an MSC graph G, which will have nodes $(Q' = \{q'_1, \ldots, q'_n\}) \cup \{start, left, right\} \cup (V = \{v_{a_1}, \ldots, v_{a_k}\})$. The edges between vertices in Q' will be identical to the transition relation δ over Q, and every node $q' \in Q'$ is labeled by an MSC with one synchronous (acknowledged) message from process P_1 to P_2, where the content of the message is $l(q)$. The vertices V will form a complete subgraph and $v_a \in V$ is labeled by an MSC where P_1 sends a (synchronously) to P_2. The start node *start* is labeled with the empty MSC. It has edges to both the *left* and *right* nodes. The node *left* is labeled by an MSC where P_3 sends the (synchronous) message "left" to P_2 and the (synchronous) message "go" to P_1. The node *right* is labeled by an MSC where P_3 sends the (synch) message "right" to P_2 and the (synch) message "go" to P_1. The *right* node has an edge to the initial state $q'_0 \in Q'$. The *left* node has edges to all vertices in V. The target nodes of G are all nodes of V as well as those nodes $q' \in Q'$ such that $q \in F$. The claim is that $L(G)$ is safely realizable iff $L(A) = \Sigma^*$. □

4 Verification

Now we turn our attention to the verification problem where the system to be verified is described by an MSC-graph G. We will consider two semantics for

the verification problem, the set $L(G)$ of all the MSCs specified by G, and the set $L^w(G)$ of all the MSCs in the weak closure. First suppose the specification is given by an automaton A accepting linearizations corresponding to "bad" behaviors. If the semantics of an MSC-graph G is $L(G)$, then the verification problem, namely, checking emptiness of $L(G) \cap L(A)$, is undecidable in general and PSPACE-complete for bounded graphs [1]. As our results will indicate, when the semantics of G is $L^w(G)$, the verification problem is undecidable even for bounded graphs. Since MSCs specify partially ordered executions, we proceed to consider partial-order specifications.

4.1 MSC Membership

Given MSC graph G and given an MSC M, we wish to know (1) is $M \in L(G)$? and (2) is $M \in L^w(G)$? There are at least two reasons to consider this problem. First, M may specify an undesirable scenario, so a positive answer to any of these two questions imply existence of a bug. Second, M may specify a desired behavior, and answering these questions can help avoid redundancy.

As discussed earlier we can equate an MSC over k processes with a "well-formed" k-tuple $\langle s_1, \ldots, s_k \rangle$ of strings s_i, where s_i indicates the linearly ordered sequence of messages sent and received by process i.

First we consider the question of checking if a given MSC M belongs to $L(G)$. There are two cases to this question depending on whether the number of processes k in the MSCs is fixed or not. We observe that for a fixed number of processes, k, the question can be answered in time $O(n^{2k})$, and we show that for an arbitrary number of processes the question is NP-complete. Boundedness is not relevant to these results.

Theorem 4. *Given an MSC-graph G and an MSC M over k processes, there is an algorithm that decides in $O(n^{2k})$ time whether $M \in L(G)$, where n is the size of the input (G, M).*

Proof. Let $M = \langle s_1, \ldots, s_k \rangle$. Let $s_i[j, j']$ denote, for $0 \leq j \leq j' \leq |s_i|$ the substring of s_i starting at position j and ending at position j'. Since k is fixed, we can build, by dynamic programming, a $2k$-dimensional array of boolean values K, where $K[d_1, \ldots, d_k, d'_1, \ldots, d'_k]$, is true for $0 \leq d_i \leq d'_i \leq |s_i|$, whenever $\langle s_1[d_1, d'_1], \ldots, s_k[d_k, d'_k] \rangle \in L(G)$. K can easily be computed in time $O(n^{2k})$. Thus, $M \in L(G)$ iff $K[0, \ldots, 0, |s_1|, \ldots, |s_k|]$ is true. □

Next we show NP-completeness for the membership problem. The proof is very similar to the proof given by [9] for "template matching" in MSC graphs, but because template matching offers more flexibility than finding a given MSC, we need a reduction from a slightly different NP-complete problem.

Theorem 5. *Given an MSC-graph G and an MSC M, it is NP-complete to determine if $M \in L(G)$, even when G is a complete graph, or when G is an acyclic graph.*

Proof. The problem is contained in NP because we can guess a path in G and easily verify that the path generates M.

To show NP-hardness, we provide a reduction from the NP-complete problem *ONE-IN-THREE-3-SAT* [11]: given a 3-CNF formula φ, is there a satisfying assignment to the variables such that each clause of φ gets *exactly one literal* assigned `true`. From a 3CNF formula $\varphi = C_1 \wedge \ldots \wedge C_m$, over variables $x_1, \ldots, x_n$, we define an MSC graph G and an MSC M over $2m + 2n$ processes. The underlying graph of G is a complete graph, and M does not depend on φ. For each clause C_j, we have two processes $P_{j,1}$ and $P_{j,2}$, and for each variable x_i, we have two processes $Q_{i,1}$ and $Q_{i,2}$. The complete graph G has $2n$ vertices $V = \{v_i, w_i \mid i \in [n]\}$, where n is the number of variables in φ. All vertices of G are initial vertices. For each i we label v_i by an MSC M_{x_i} in which there is a one message (labeled, say, a) from process $P_{j,1}$ to $P_{j,2}$ precisely when variable x_i appears positively in C_j. In addition, there is a message a sent from $Q_{i,1}$ to $Q_{i,2}$ in M_{x_i}. Likewise, w_i is labeled by an MSC $M_{\bar{x_i}}$, which does the opposite of M_{v_i}: there is one message labeled a from process $P_{j,1}$ to $P_{j,2}$ when variable x_i appears negatively. Again, in addition, the message a is sent from $Q_{i,1}$ to $Q_{i,2}$.

Finally, we define M, which does not depend on φ. In M, for each j, there is one a-message sent from $P_{j,1}$ to $P_{j,2}$, and for each i there is an a-message sent from $Q_{i,1}$ to $Q_{i,2}$. It is not difficult to see that $M \in L(G)$ iff there is a satisfying assignment to φ that sets precisely one literal in each clause to true. □

Now we consider the membership question for weak-closure semantics: is $M \in L^w(G)$? This problem turns out to be much easier:

Theorem 6. *Given an MSC-graph G and an MSC M, there is an algorithm that in time $O(|G||M|)$ determines whether $M \in L^w(G)$.*

Proof. Suppose G and $M = \langle s_1, \ldots, s_k \rangle$ are defined over k processes. For each process i, let G_i be the "projection" of G onto the events of process i: G_i is like G, but each vertex v in G_i is labeled with the projection onto process i of the MSC labeling v_j in G. G_i can be viewed as an ordinary automaton over the alphabet of events belonging to process i. Then $M \in L^w(G)$ iff $s_i \in G_i$ for each i. Building G_i's can be done in linear time, and checking whether $s_i \in G_i$ can be done in time $O(|G_i||s_i|)$, for each $i \in [k]$. Thus, the total time is $O(|G||M|)$. □

4.2 Checking Local Properties

Given G, we want to know whether $L^w(G)$ satisfies a property φ. A property φ is linearization independent if it holds for one linearization of an MSC iff it holds for all. A property φ of MSCs is said to be *local* if it (syntactically) only describes events on one process. Boolean combinations of local properties are clearly linearization independent. It is known that checking linearization independent properties of $L(G)$ can be done in polynomial-time [10]. However, the following theorems shows that we cannot hope to find efficient model checking algorithms for $L^w(G)$.

Theorem 7. *There are local properties φ_1 and φ_2 such that for a finite MSC set L, it is coNP-complete to determine if every MSC in L^w satisfies $\varphi_1 \vee \varphi_2$.*

Proof. The membership in co-NP is obvious. The hardness proof is a reduction from 3SAT. Let $\Gamma = \langle C_1, \ldots, C_m \rangle$ be the clauses of the 3SAT formula, ordered

in some arbitrary way, and let $x_1, \ldots, x_n$ be its variables. We will add new variables y and $z_1, \ldots, z_m$. Our new ordered list of clauses will be $\Delta = \langle y \vee \neg z_1, y \vee \neg z_2, \ldots, y \vee \neg z_m, C_1 \vee z_1, C_2 \vee z_2, \ldots, C_m \vee z_m \rangle$. Clearly, Γ has a satisfying assignment iff Δ has a satisfying assignment with $y = 0$. Let $\Delta|_1^k$ denote the first k clauses in the list Δ. Notice that for every clause C_i and for every assignment to variables occurring in C_i, there is a satisfying assignment of $\Delta|_1^{i+m-1}$ which agrees with that assignment.

Now we are ready to describe our MSC set, and our properties φ_1 and φ_2. The MSC set will consist of one process for every variable and every clause in Δ. In addition, there will be an extra process called P_f, which will serve to tabulate whether the formula has been satisfied or not. There will be one MSC, M_t based on the "trivial" satisfying assignment to Γ, namely P_y will send *true* to every clause that contains it, in their lexicographical order. Likewise, the P_{z_i}'s will send *true* to every clause that contains z_i, respectively. The x_i variables, can either send *true* or *false* to their clauses, it doesn't matter here. Each clause process $P_{C'}$, after receiving its truth assignment in messages (which it reads in the lexicographical order of the variables), then receives a message from its predecessor clause (if there is one) which either indicates that the prior clauses have all been satisfied or not. If the prior clauses have been satisfied, and if C' itself has also been satisfied, then C' propagates the "satisfied" message to the next clause in the ordered list. Otherwise, it propagates "not satisfied". The last clause C'' propagates this message to P_f, which does nothing other than to receive it. Clearly, for the assignment on which M_t is based, P_f will receive a satisfied message.

Next, for each clause $C_i \vee z_i$, and for each satisfying assignment ρ to the variables in $C_i \vee z_i$ (there are only a constant number of these, since C_i is a 3CNF clause), we will add a new MSC $M_{C_i,\rho}$ which mimics the same thing as above, only the assignment to the variables is one consistent with both ρ and the assignment mentioned above which satisfies $\Gamma|_1^{i+m-1}$. Finally, we add another MSC M_y, whose only purpose is to exhibit one MSC such that the "assignment" to y is *false*.

Our set L of MSCs contains M_t, M_y and the MSCs $M_{C_i,\rho}$. Consider an MSC $M \in L^w$. If P_f receives a "satisfied" message, then we can construct a satisfying assignment for Δ from M. Moreover, if P_y sends *false*'s in M, then we can construct a satisfying assignment which assigns $y = 0$, i.e., a satisfying assingment to Γ. We claim that the converse holds as well, i.e., if there is such a satisfying assignment, then there will be an M weakly-implied by L where P_y sends *false* (call this $\neg\varphi_1$) and P_f receives "satisfied" (call this $\neg\varphi_2$). □

It follows that checking whether every MSC in $L^w(G)$, for an acyclic MSC-graph G, satisfies a boolean combination of local properties is coNP-complete.

Theorem 8. *There is a boolean combination φ of local properties, such that given a bounded MSC-graph G, it is undecidable to check if every MSC in $L^w(G)$ satisfies φ.*

Proof. The proof uses precisely the same complete MSC graphs given in the proof of Theorem 1, which reduce an instance of RPCP to checking whether

$L(G) = L^w(G)$. Note that in that setting there is an implied but unspecified MSC (and thus a solution to the RPCP) if and only if every message sent by process P_1 to P_2 is of the form $(i, 0)$, while every message sent by P_3 to P_4 is of the form $(i, 1)$, or vice versa. □

Acknowledgements. This research was partially supported by NSF Career award CCR97-34115, NSF award CCR99-70925, and Sloan Faculty Fellowship.

References

[1] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *Proc. of 22nd Int. Conf. on Software Engineering*, 2000.

[2] R. Alur, G.J. Holzmann, and D. Peled. An analyzer for message sequence charts. *Software Concepts and Tools*, 17(2):70–77, 1996.

[3] R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Concurrency Theory, Tenth Intl. Conference*, LNCS 1664, pages 114–129, 1999.

[4] H. Ben-Abdallah and S. Leue. MESA: Support for scenario-based design of concurrent systems. In *Proc. 4th TACAS*, LNCS 1384, pages 118–135, 1998.

[5] G. Booch, I. Jacobson, and J. Rumbaugh. *Unified Modeling Language User Guide*. Addison Wesley, 1997.

[6] H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In *Proc. 2nd TACAS*, 1997.

[7] J. Henriksen, M. Mukund, K. Narayan Kumar, and P.S. Thiagarajan. On message sequence graphs and finitely generated regular MSC languages. In *Proc. 27th ICALP*, LNCS 1853, pp. 675–686, 2000.

[8] G.J. Holzmann, D.A. Peled, and M.H. Redberg. Design tools for for requirements engineering. *Lucent Bell Labs Technical Journal*, 2(1):86–95, 1997.

[9] A. Muscholl, D. Peled, and Z. Su. Deciding properties of message sequence charts. In *Foundations of Software Science and Computation Structures*, 1998.

[10] D. Peled. Specification and verification of message sequence charts. In *Proc. IFIP FORTE/PSTV*, 2000.

[11] T.J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th ACM Symp. on Theory of Computing*, pages 216–226, 1978.

[12] ITU-T recommendation Z.120. Message Sequence Charts (MSC'96), 1996.

Reasoning about Sequential and Branching Behaviours of Message Sequence Graphs

P. Madhusudan

Institute of Mathematical Sciences, C.I.T. Campus, Taramani, Chennai, India.
madhu@imsc.ernet.in

Abstract. We study the model-checking problem of message-sequence graphs (MSGs). In the sequential setting, we consider the set of message-sequence charts (MSCs) represented by an MSG and tackle specifications given in monadic second-order logic. We show that this problem, without any restrictions on the MSGs, is decidable. We then turn to branching behaviours of MSGs, define a notion of an unfolding of an MSG, and show that the model-checking problem on unfoldings is also decidable. Our results are stronger and imply that, over an appropriate universe, satisfiability and synthesis of MSCs and MSGs, respectively, are decidable.

1 Introduction

Message sequence charts (MSC) are a popular visual formalism used to describe design requirements of concurrent message-passing systems [16,10]. They are used to describe behaviours of systems in early models of system design. An MSC describes a single partially-ordered execution of the system and a collection of these diagrams formalize the set of possible scenarios exhibited by a model. These diagrams are usually used in the design of distributed telecommunication protocols and are also known as timing-sequence diagrams or message-flow diagrams [16].

Message sequence graphs (MSGs), also known as MSC-graphs, are a convenient mechanism to specify an infinite collection of MSCs. These graphs basically have a finite set of atomic MSCs and the MSG defines some (regular) ways to combine them using concatenation, choice, and repetition. As a model of concurrent systems they are interesting as they exhibit a sequential composition of parallel behaviours, as opposed to most other models which are parallel compositions of sequential machines.

Since MSCs and MSGs have a formal semantics, they are amenable to analysis. The issue of *model-checking* MSCs and MSCs defined by an MSG has been an area of active study in recent years [1,11,14,13,2,9]. Automated verification of MSGs could enable engineers to check whether the design requirements have been met and whether it describes the set of desirable (or undesirable) behaviours correctly.

Alur and Yannakakis [2] give a detailed account of the model-checking problem for MSGs and the corresponding complexity involved. They consider the

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 809–820, 2001.

problem of deciding whether the *linearizations* of events of all MSCs defined by an MSG satisfy some linear-time property. As they show, the problem is undecidable even if the specification of good behaviours is given by a deterministic finite automaton. Consequently, in [2] the authors consider a syntactically identifiable sub-class of MSGs called *bounded MSGs*, for which they prove that the linearizations form a regular set of sequences, and thereby solve the model-checking problem for this class.

In [7,9], the authors define a notion of *regular* MSC-languages which are recognizable by distributed communicating finite-state devices and study their properties. The global device is forced to be finite-state by constraining channels to have a bounded capacity. However, this class of languages are not contained in, nor contain, the class of languages defined by MSGs [8]. It is also shown that bounded MSGs define only regular MSC-languages and hence it follows from their results that one can decide specifications written in MSO on such bounded MSGs.

In this paper, the main point of departure from previous work is to completely ignore linearizations of events of an MSC and instead work directly on the graphical (visual) description of the charts. The main result we show is that the problem of deciding whether the set of all MSCs defined by an MSG satisfy a property given as a MSO formula is decidable. The solution does not assume *any* restriction on the atomic MSCs— they can exhibit non-FIFO behaviour, the messages can have labels, processes can have local actions, etc. Also, the MSGs are not restricted in any way— they can be unbounded in the sense of [2].

In the second half of our paper, we show how to handle branching-time behaviours of MSGs. We define the notion of an *unfolding* of an MSG which we believe is useful to reason about its behaviours, and then show that this infinite graph has a decidable monadic theory. This then enables the verification of properties such as asking whether every (finite) MSC defined by a given MSG is always extendable to another defined by the MSG, which satisfies a property expressed in MSO.

Our results are in fact are stronger. Let us fix any finite set of atomic finite MSCs and consider the universe of all MSCs generated by this set using concatenation. Then we show that a restricted form of satisfiability— whether there is an MSC which is formed using the atomic MSCs which satisfies a given formula— is decidable. We also show that the problem of synthesis of MSGs— whether there is an MSG whose unfolding satisfies a given specification— is decidable.

Our results and the simplicity of the concepts in our proofs suggest that *structural* logics are perhaps more amenable to reasoning about MSCs than logics which specify properties of the linearizations of an MSC. Logics such as monadic second-order logic, or an adaptation of the logic LTrL [19], or *localised logics* such as mLTL [15] can all be seen as structural logics which when defined over MSCs appropriately will be decidable.

Turning to the theoretical interest of our results, the study of infinite graphs which have a decidable monadic theory is an interesting area of research [5,

20,18]. Our results exhibit a class of infinite graphs, which arise naturally in the study of concurrent systems, that do have a decidable monadic theory. A careful reader will note that our proofs do not assume much about MSCs and the results can be extended to the setting where we work with any set of atomic finite n-boundaried graphs. Also, our results are stronger as we show that we can work over an infinite universe of infinite graphs and the monadic second-order formulas describe only "regular" subsets of this class. We can hence answer questions like whether there is a graph in such a class which satisfies a property given in MSO.

In the next section we introduce our problem setting and deal with the sequential behaviours of MSGs in Section 3. In Section 4 we define the notion of an unfolding of an MSG and show how we can reason about it. Due to space restrictions we provide only the main ideas— more details can be found in [12].

2 Preliminaries

A message sequence chart (MSC) is a diagram which depicts a single partially-ordered execution of a distributed message-passing system. For example, consider the MSC depicted on the left-hand side of Figure 1. It consists of four processes— a customer c, a manager m, a database server d and a security checker s. The MSC depicts a particular scenario where the customer requests for data with his identity. Though the customer sends the request first followed by the id, the messages get delivered in reverse order. After receiving the id, the manager sends it to the security checker for verification. It also sends the request to the database. After receiving messages from them, the manager serves the customer with the data. Note that the security process has a local action where it checks the id. The MSC represents one partially-ordered behaviour of the way the messages were sent and received— some events are ordered while others, like m receiving the request and s receiving the id, are not.

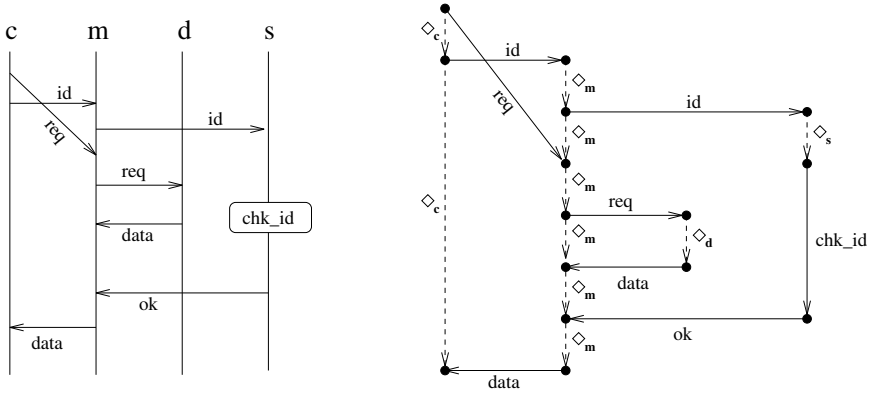


Fig. 1. An MSC and its associated graph

We work with MSCs by considering them as labelled directed graphs. For example, Figure 1 illustrates an MSC and the corresponding graph representation for it. The graph representing the MSC has vertices at points where the events take place, and edges denote messages as well as local actions. A vertex with an outgoing message edge is to be viewed as the action of sending a message, while the vertex at the other end is seen as the point where the message is received. The vertices of each process p are connected in the natural way according to the local linear flow of time, and these edges are labelled by a special symbol $\diamond_p$ (denoted by dotted lines).

Let us now define MSCs formally. Let us fix a finite set of processes $\mathcal{P}$ and let p, q, r range over $\mathcal{P}$. Let us also fix a finite set of local actions Γ_l and a set of messages Γ_m. For each $p \in \mathcal{P}$, let us also have a *next-state symbol* $\diamond_p$ and let $\Gamma_n = \{\diamond_p \mid p \in \mathcal{P}\}$. We fix all these for the rest of the paper and assume that all these sets are pairwise disjoint. Let $\Gamma = \Gamma_l \cup \Gamma_m \cup \Gamma_n$. In the above example, $\mathcal{P} = \{c, m, d, s\}$, $\Gamma_m = \{id, req, data, ok\}$, $\Gamma_l = \{chk\_id\}$, $\Gamma_n = \{\diamond_c, \diamond_m, \diamond_d, \diamond_s\}$.

Let us fix some notations on directed graphs. A directed graph G over a finite label-set L is a tuple (V, E) where V is a set of vertices and $E \subseteq V \times L \times V$. By $v \xrightarrow{l} v'$ we mean $(v, l, v') \in E$. For a set $L' \subseteq L$ and $v \in V$, we denote by $out_{L'}(v)$ the set of all vertices such that there is an L'-labelled edge from v to it— i.e. $out_{L'}(v) = \{v' \in V \mid \exists l \in L' : (v, l, v') \in E\}$. Similarly, $in_{L'}(v)$ denotes the set of vertices which have an L'-labelled edge to v.

For a linear-order $\leq$ on a finite set S, let $\lessdot$ denote the *covering relation for* $\leq$: for any $x, y \in S$, $x \lessdot y$ iff ($x \leq y$ and $\forall z \in S$, if $x \leq z \leq y$, then $z = x$ or $z = y$). Also, for any $n \in \mathbb{N}$, let $[n]$ denote the set $\{1, \ldots, n\}$.

Definition 1. *A message sequence chart (MSC) is a tuple* $(\{V_p\}_{p \in \mathcal{P}}, E)$ *where, if* $V = \bigcup_{p \in \mathcal{P}} V_p$, *then:*

- V *is finite*
- $E \subseteq [\bigcup_{p \in \mathcal{P}} (V_p \times (\Gamma_l \cup \Gamma_n) \times V_p)] \ \cup \ [\bigcup_{p,q \in \mathcal{P}, p \neq q} (V_p \times \Gamma_m \times V_q)]$
- *For each* $p \in \mathcal{P}$, *there is a linear-order* $\leq_p$ *on* V_p *such that* $\forall v, v' \in V_p$: $(v \lessdot v'$ *iff* $\exists l \in \Gamma_l \cup \Gamma_n : v \xrightarrow{l} v')$.
- $out_{\Gamma_l \cup \Gamma_n}(v)$ *is a singleton set.*
- *For each* $v \in V$, *one of the three sets* $out_{\Gamma_l}(v)$, $out_{\Gamma_m}(v)$ *and* $in_{\Gamma_m}(v)$ *is a singleton set and the other two are empty.*
- $G = (V, E)$ *is acyclic.* □

The second condition demands that the edges within a process be labelled by local actions or next-state symbols while edges between different processes are labelled using the message alphabet. The next two conditions demand that the events of each local process are totally ordered and are connected in this order using exactly one edge. We also demand that at each point, exactly one of the following happen: a message is received, a message is sent, or there is a local action. We finally require the events to be partially ordered.

Though we do not require it, for technical ease let us assume that each V_p is nonempty. For an MSC m and a process $p \in \mathcal{P}$, the linear order on V_p satisfying the condition above is clearly unique, and we denote this by $\leq_p$. Let

the minimum vertex of V_p in m under this ordering be denoted by $first_p^m$ and the maximum vertex by $last_p^m$. We identify an MSC with its equivalence class with respect to isomorphism— i.e. two graphs will represent the same MSC if they are isomorphic.

Concatenation of two MSCs is done by (asynchronously) concatenating the individual process evolutions of the MSCs. For example, Figure 2 illustrates the concatenation of two MSCs. Formally,

Definition 2. *Let $m_1 = (\{U_p\}_{p\in\mathcal{P}}, E_1)$ and $m_2 = (\{V_p\}_{p\in\mathcal{P}}, E_2)$. Let U_p and V_p be disjoint sets (if they are not, relabel vertices to make them disjoint). The concatenation of m_1 and m_2, denoted by $m_1 \cdot m_2$ is the MSC $(\{W_p\}_{p\in\mathcal{P}}, E)$ where $W_p = U_p \cup V_p$ and $E = E_1 \cup E_2 \cup \{(last_p^{m_1}, \diamond_p, first_p^{m_2}) \mid p \in \mathcal{P}\}$.* □

Note that concatenation is associative.

We now fix notations for talking about finite-automata. A deterministic finite-state automaton (DFA) $\mathcal{A}$ is a structure $(\Sigma, Q, q_{in}, \delta, F)$, where Σ is a finite alphabet, Q is a finite set of states, $q_{in} \in Q$ is the initial state, $\delta : Q \times \Sigma \to Q$ is the transition function and $F \subseteq Q$ is the set of final states. The transition function δ can be extended to a function $\delta' : Q \times \Sigma^* \to Q$ by defining $\delta'(q, a) = \delta(q, a)$ and $\delta'(q, x \cdot a) = \delta(\delta'(q, x), a)$. We say that $\mathcal{A}$ accepts a word $x \in \Sigma^*$ if $\delta'(q_{in}, x) \in F$. The language of $\mathcal{A}$, denoted by $L(\mathcal{A})$ is the set of strings it accepts.

Message sequence graphs are a convenient mechanism to define a collection of MSCs. We use a slightly different but equivalent terminology than usually found in the literature.

Definition 3. *A message sequence graph (MSG) is a structure $\mathcal{M} = (\Sigma, \mathcal{A}, M, h)$ where Σ is a finite alphabet, $\mathcal{A}$ is a finite automaton on Σ, M is a finite collection of MSCs over $\mathcal{P}$ and $h : \Sigma \to M$ is a bijection that identifies an MSC in M for each symbol in Σ.* □

We specify properties of MSCs as well as other directed graphs using *monadic second-order logic* (MSO). Let Π be any finite alphabet. Then the monadic second-order logic over directed graphs with edge labels from Π is defined as follows: let there be an infinite number of first-order variables $\{x, y, z, \ldots\}$ and second-order variables $\{X, Y, Z, \ldots\}$. The set of MSO formulas is the smallest set containing:

- The atomic formulas of the kind $x \in X$ and $x \xrightarrow{l} y$ where $l \in \Pi$.
- The formulas $\varphi \vee \psi$, $\neg\varphi$, $\exists x\varphi(x)$ and $\exists X\varphi(X)$, where φ and ψ are MSO formulas.

  $\varphi(x)$ ($\varphi(X)$) denotes a formula with free variable x (X).

Let $G = (V, E)$ be a (finite or infinite) graph where $E \subseteq V \times \Pi \times V$ and let φ be a formula. An *interpretation* for φ is a function I which assigns to each free first-order variable of φ a vertex in V and assigns to each free second-order variable of φ a subset of V. The semantics of a formula being true under an interpretation I is the usual one. A sentence is a formula φ without free

variables. For a sentence φ and a graph G, we say G *satisfies* φ if φ is true in G and we denote this by $G \models \varphi$.

In the proofs, we work with a different set of formulas that is as expressive, but which has no formulas with first-order variables (as done in [18]). This restricted syntax has atomic formulas $X \subseteq Y$, $Singleton(X)$ and $X \xrightarrow{l} Y$. Other formulas are $\varphi \vee \psi$, $\neg \varphi$ and $\exists X \varphi(X)$. It is easy to see that this syntax is exactly as expressive as the original one.

3 Sequential Behaviours of MSGs

Let us fix an MSG $\mathcal{M} = (\Sigma, \mathcal{A}, M, h)$ for the rest of this section. An MSC m is generated by $\mathcal{M}$ if there is a string $x \in \Sigma^*$ accepted by $\mathcal{A}$ such that the concatenation of the MSCs in x is m.

Formally, for $x = a_0 a_1 \dots a_n \in \Sigma^*$, let $msc_{M,h}(x) = h(a_0) \cdot h(a_1) \dots h(a_n)$. When M and h are clear from context, we denote $msc_{M,h}(x)$ by $msc(x)$. The *MSC-language of* $\mathcal{M}$ is $\mathcal{L}(\mathcal{M}) = \{msc(x) \mid x \in L(\mathcal{A})\}$. We say that $\mathcal{M}$ generates m if $m \in \mathcal{L}(\mathcal{M})$.

The sequential model-checking problem for MSGs is then the following: Given an MSO sentence φ over directed graphs labelled by Γ and an MSG $\mathcal{M}$, do all the MSCs generated by $\mathcal{M}$ satisfy φ?

MSO is an expressive mechanism to state the required structural properties of MSCs. For example, we can state in the context of the example in Figure 1 that if the manager sends the data, then there must be a corresponding confirmation of id from the security checker. It can also express a variety of safety and liveness conditions (like "if p sends a request, then it eventually gets an acknowledgement", "if the messages are delivered in FIFO fashion, then a property φ holds", etc.).

The main lemma we use to prove that the above problem is decidable is to show that for any property φ, the strings $x \in \Sigma^*$ such that $msc_{M,h}(x) \models \varphi$ form a regular subset of Σ^*. The proof follows the structure of Thomas' adaptation of the proof by Büchi [3] and Elgot [6] that classes of languages defined by monadic second-order formulas over finite strings correspond to the class of regular languages (see [17,18]). This proof works by inductively associating with every formula $\varphi(X_1, \dots, X_k)$ an automaton $\mathcal{A}_\varphi$ over words that encode interpretations of the variables over the structure and accept the word iff the structure, under the interpretation, satisfies φ.

Let φ be a MSO formula over MSCs. The idea is to augment the alphabet Σ to get a new alphabet which has letters of the form (a, I), where $a \in \Sigma$. Let x be a finite string over this alphabet, say $x = (a_0, I_0), \dots (a_n, I_n)$. The MSC we are working on now is $m = msc_{M,h}(a_0 \dots a_n)$. The second component of the alphabet will define an interpretation of the free variables of φ over the MSC m. The idea is that I_j will encode an interpretation of the variables over the MSC $h(a_j)$, for each $j \in \{0, \dots, n\}$. Together, the I_j's will define a complete interpretation of the variables over the graph m.

We then follow the automata-theoretic approach to show inductively that for every formula φ, there is an automaton over a suitable alphabet which accepts a word iff the MSC defined by the word, under the interpretation of the free variables of φ over this MSC, satisfies the formula φ.

Now, let φ have k free (second-order) variables. For a letter $a \in \Sigma$, we require a way to encode all interpretations of the k variables over the MSC $h(a)$. Let $h(a)$ have r vertices. Then a 0-1 matrix Z with k rows and r columns can represent any such interpretation: the j^{th} vertex of $h(a)$ (according to some fixed ordering) belongs to the i^{th} variable iff the entry in Z on the i^{th} row and j^{th} column is 1.

Let us fix some notations for matrices. For any $k, r \in \mathbb{N}$, let $S_{k\times r}$ denote the set of all 0-1 matrices with k-rows and r-columns. For $Z \in S_{k\times r}$, we denote by $Z[i,j]$ $(i \in [k],\, j \in [r])$, the element at the i^{th} row and j^{th} column in Z.

Lemma 1. *Let φ be an MSO property on directed graphs with edge labels from Γ. Let M be a finite set of MSCs, Σ be a finite alphabet and let $h : \Sigma \to M$ be a bijection. Let $L_\varphi^{\Sigma,M,h} = \{x \in \Sigma^* \mid msc_{M,h}(x) \models \varphi\}$. Then $L_\varphi^{\Sigma,M,h}$ is a regular subset of Σ^*. Moreover, one can effectively construct a finite-state automaton accepting $L_\varphi^{\Sigma,M,h}$.*

Proof: For each MSC $m = (\{V_p\}_{p\in\mathcal{P}}, E) \in M$, let $s(m)$ denote the number of vertices in m, i.e. $|V|$, where $V = \bigcup_{p\in\mathcal{P}} V_p$. Also fix some ordering on V.

For any $k \in \mathbb{N}$, let $\Sigma_k = \{(a, Z) \mid a \in \Sigma, Z \in S_{k\times s(h(a))}\}$. A letter $(a, Z) \in \Sigma_k$ defines an interpretation of k variables on subsets of vertices of $h(a)$: a vertex u of the MSC $h(a)$ is in X_i iff u is the j^{th} element of m and $Z[i,j] = 1$.

A string $x = (a_0, Z_0)\ldots(a_n, Z_n) \in \Sigma_k^*$ encodes an interpretation of the k variables on the MSC $h(a_0) \cdot h(a_1)\ldots h(a_n)$— this is given by taking the union of the individual interpretations. The claim we prove now is:

Claim For each MSO formula $\varphi(X_1, \ldots X_k)$ there is a deterministic finite-state automaton $\mathcal{A}_\varphi$ over the alphabet Σ_k which accepts a string x iff the MSC defined by x, under the interpretation defined by x, satisfies φ.

Proof of Claim We show the claim by induction on the structure of φ. It is easy to see that the atomic formulas $X \subseteq Y$ and $Singleton(X)$ can be checked by an automaton. For the atomic formula $X \xrightarrow{l} Y$ where $l \in \Gamma$, the automaton first checks whether the interpretations for X and Y are singleton sets and rejects the word if it is not. Let X be interpreted as $\{u\}$ and Y as $\{v\}$. If u and v belong to the same atomic MSC $h(a)$ for some (a, Z) in the input sequence, then clearly $u \xrightarrow{l} v$ iff $u \xrightarrow{l} v$ in $h(a)$. The automaton can decide this by a table-lookup and accept or reject the word accordingly.

We are left with the case where u and v belong to different atomic MSCs. If $l \in \Gamma_l \cup \Gamma_m$, then the automaton rejects. Otherwise let $l = \diamond_p$. It checks now whether u and v belong to consecutive atomic MSCs m, m' in the input, and whether u is $last_p^m$ and v is $first_p^{m'}$. If so, it accepts the word and otherwise rejects it.

The other formulas are easy to tackle— disjunction, negation and existential quantification are handled by using the fact that DFAs are effectively closed under union, complement and projection. **End of claim**

From the Claim, it is clear that for any sentence φ, we can construct an automaton over Σ^* such that the automaton accepts a word x iff $msc_{M,h}(x) \models \varphi$. □
We can now solve the model-checking problem for MSGs:

Theorem 1. *Given an MSG $\mathcal{M}$ and an MSO formula φ over MSCs, the problem of checking whether all MSCs generated by $\mathcal{M}$ satisfy φ is decidable.*

Proof: Let the MSG be $\mathcal{M} = (\Sigma, \mathcal{A}, M, h)$. Construct using Lemma 1, an automaton accepting $L_{\neg\varphi}^{\Sigma,M,h}$. This language consists of all sequences $x \in \Sigma^*$ such that $msc(x) \not\models \varphi$. Take its intersection with $\mathcal{A}$. Clearly, the intersection is empty iff all the MSCs generated by $\mathcal{M}$ satisfy φ. □

Another consequence of Lemma 1 is that we can solve a related satisfiability problem:

Theorem 2. *Let M be a finite set of MSCs and φ be an MSO formula over MSCs. Then the problem of finding whether there is an MSC m, which can be expressed as a concatenation of MSCs in M, such that m satisfies φ, is decidable.*

Proof: Take an alphabet Σ, with $|\Sigma| = |M|$, take a 1-1 function $h : \Sigma \to M$, and construct, using Lemma 1, an automaton accepting the language $L_{\varphi}^{\Sigma,M,h}$. Clearly, there is an MSC m formed using MSCs in M which satisfies φ iff $L_{\varphi}^{\Sigma,M,h} \neq \emptyset$. □

Observe that all the above results can be adapted to infinite concatenations as well. We can easily extend Definition 2 to define infinite concatenations of finite atomic MSCs in the natural way. We can extend the notion of an MSG such that the automaton provided accepts regular ω-languages over Σ. Then Lemma 1 smoothly extends to MSO formulas over infinite MSCs generated by the finite collection of finite atomic MSCs in the MSG. All we need to do is to work over *nondeterministic Büchi automata* [4,17,18] and use the property that the languages these automata accept are effectively closed under union, complement and projection. The analogous model-checking and satisfiability theorems for infinite MSCs will hold as well. Such an extension will facilitate formulating liveness properties of the system.

4 Branching Behaviours of MSGs

In this section, our main goal is to define the unfolding of an MSG and show that its MSO theory is decidable. However, as in the previous section, we prove a stronger theorem which says that the set of all unfoldings which satisfy a formula is regular. Let us prove this lemma before we tackle MSGs.

Fix $D \in \mathbb{N}$ and a finite alphabet Σ. A *Σ-labelled D-ary tree* is a tuple (T, ρ) where T is the graph $G = (V_T, E_T)$ where $V_T = [D]^*$, $E_T = \{(x, x \cdot d) \mid x \in V_T, d \in [D]\}$ and $\rho : [D]^+ \to \Sigma$. (Note that the root is not labelled). Let M be a finite collection of (finite) MSCs and $h : \Sigma \to M$ be a bijection.

Definition 4 (Wrap of an MSC). *Let $a \in \Sigma$, $m = h(a) = (\{V_p\}_{p\in\mathcal{P}}, E)$. Then the* wrap of m *is the graph $G_w = (V_w, E_w)$ where $V_w = (\bigcup_{p\in\mathcal{P}} V_p) \cup \{f_a, l_a\}$ and $E_w = E \cup \{(f_a, *_a, first_p^m), (last_p^m, *_a, l_a) \mid p \in \mathcal{P}\}$.* □

Note that the wrap of an MSC is just the MSC with two extra vertices, one of which is connected to the first nodes of the MSC and the last nodes of the MSC are connected to the other. Now we associate with any Σ-labelled D-ary tree (T, ρ), the graph obtained by replacing each vertex u of T by the wrap of $h(\rho(u))$.

Definition 5. *Let (T, ρ) be a Σ-labelled D-ary tree with $T = ([D]^*, E_T)$. Then the graph associated with (T, ρ) is the graph $G = (V, E)$ where $V = \{\langle u, v\rangle \mid u \in [D]^+$ and v is a vertex in $wrap(h(\rho(u)))\} \cup \{r\}$ and*

$$E = \{(\langle u, v\rangle, l, \langle u, v'\rangle) \mid v \xrightarrow{l} v' \text{ in } wrap(h(\rho(u)))\} \cup$$
$$\{(\langle u, l_a\rangle, *, \langle u', f_b\rangle) \mid u \to u' \text{ in } T \text{ and } \rho(u) = a, \rho(u') = b\} \cup$$
$$\{(r, *, \langle i, f_a\rangle) \mid i \in [D], \rho(i) = a\}.$$ □

Let $\Pi = \Gamma \cup \{*_a \mid a \in \Sigma\} \cup \{*\}$. We now work with MSO formulas over graphs labelled over Π. Consider such a formula φ. We show now that the set of Σ-labelled D-ary trees T such that the graph represented by T satisfies φ, is a regular tree-language (of Σ-labelled infinite trees).

The idea behind the proof is exactly the same as in the sequential setting. We extend the alphabet so that each symbol can now encode an interpretation of variables over the (wrap of the) MSC at the node. Then, we can show that with every formula $\varphi(X_1, \ldots, X_k)$, one can associate a *parity tree automaton* which accepts a tree iff the graph associated with the tree along with the interpretation of $X_1, \ldots, X_k$ coded in it, satisfies φ. (We do not define parity tree automata as we will not give proofs here— see [17,18] for details.) We then have:

Lemma 2. *Let φ be an MSO property on directed-graphs with edge-labels from Π. Let $D \in \mathbb{N}$, M be a finite set of MSCs, Σ be a finite alphabet and let $h : \Sigma \to M$ be a bijection. Let $\mathcal{L}_\varphi^{\Sigma,M,h}$ be the set of all Σ-labelled D-ary trees (T, ρ) such that the graph associated with it satisfies φ. Then $\mathcal{L}_\varphi^{\Sigma,M,h}$ is a regular tree-language. Moreover, one can effectively construct a finite-state parity tree-automaton accepting $\mathcal{L}_\varphi^{\Sigma,M,h}$.* □

We do not give the proof here as the ideas are exactly the same as in the sequential case and the rest is only detail.

Let us now turn to MSGs. An MSG in this section will be a tuple $\mathcal{M} = (\Sigma, TS, M, h)$ where Σ, M and h are as before and where TS is a (finite) transition system (Q, q_{in}, δ) where Q is a finite set of states, $q_{in} \in Q$ is the initial state and $\delta \subseteq Q \times \Sigma \times Q$. Let us define the successor function as $succ(q) = \{(a, q') \mid (q, a, q') \in \delta\}$ for each $q \in Q$. For technical simplicity, let us assume that the number of successors for each state is the same— let $\forall q \in Q : |succ(q)| = D$, for some $D \in \mathbb{N}$. Let us also order the successors of each state and let $succ(q)[i]$ denote the i^{th} successor of q, where $i \in [D]$.

Fix an MSG $\mathcal{M} = (\Sigma, TS, M, h)$ for the rest of this section. The $(\Sigma \times Q)$-*unfolding* of $\mathcal{M}$ is the $(\Sigma \times Q)$-labelled D-ary tree (T, ρ) where $T = (V_T, E_T)$ and ρ is the unique labelling which satisfies:

- $\rho(i) = succ(q_{in})[i]$, for every $i \in [D]$
- If $\rho(x) = q$, then $\rho(x \cdot i) = succ(q)[i]$, for every $i \in [D], x \in V_T$.

The Σ-unfolding of $\mathcal{M}$ is obtained by projecting the labels of the above tree to the first component. Now the *unfolding of the MSG M* is the graph associated with the Σ-unfolding of $\mathcal{M}$. [1]

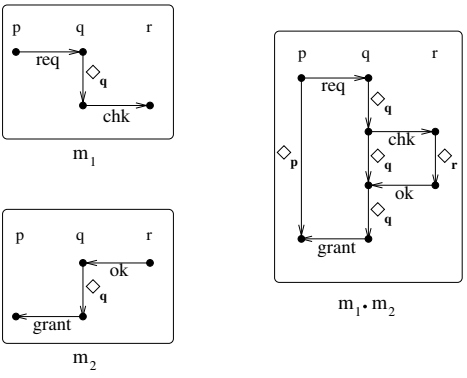


Fig. 2. Concatenation of MSCs

For example, for $\Sigma = \{a, b, c\}$ with $h(a) = m_1$ and $h(b) = m_2$, where m_1 and m_2 are the MSCs depicted in Figure 2, Figure 3 illustrates a transition system, part of its Σ-unfolding and the corresponding unfolding of the MSG.

Note that the notion of an unfolding does *not* imply that we have synchronous concatenation of MSCs. Indeed, as in the previous section, we can in the specification dynamically interpret the edges required for asynchronous concatenation and MSO formulas can then reason about the MSCs thus generated. The reason we use the above definition is to enable us to clearly describe the points at which the branching of behaviours takes place.

We can now show that the branching-time model-checking problem for MSGs is decidable:

Theorem 3. *Let $\mathcal{M}$ be an MSG and let φ be a MSO property on graphs labelled over Π. Then the problem of checking whether the unfolding of $\mathcal{M}$ satisfies φ is decidable.*

Proof: First construct, using Lemma 2, a parity tree-automaton $\mathcal{A}_\varphi$ that accepts exactly the Σ-labelled D-ary trees such that the associated graph satisfies φ. Now, TS can be converted to a tree automaton $\mathcal{A}_\mathcal{M}$ accepting the language consisting of a single tree which is the Σ-unfolding of $\mathcal{M}$. Now, clearly, there is a tree accepted by both $\mathcal{A}_\varphi$ and $\mathcal{A}_\mathcal{M}$ iff the unfolding of $\mathcal{M}$ satisfies φ. □

We can now show that various branching-time properties of MSGs, like one saying that every finite MSC generated by the MSG is extendable to another generated by the MSG which satisfies a property φ on MSCs, is decidable. We can also prove an associated *synthesis* problem, which is an analogue of Theorem 2:

[1] Note that the unfolding of the MSG is independent of the way the successors of states of TS were ordered.

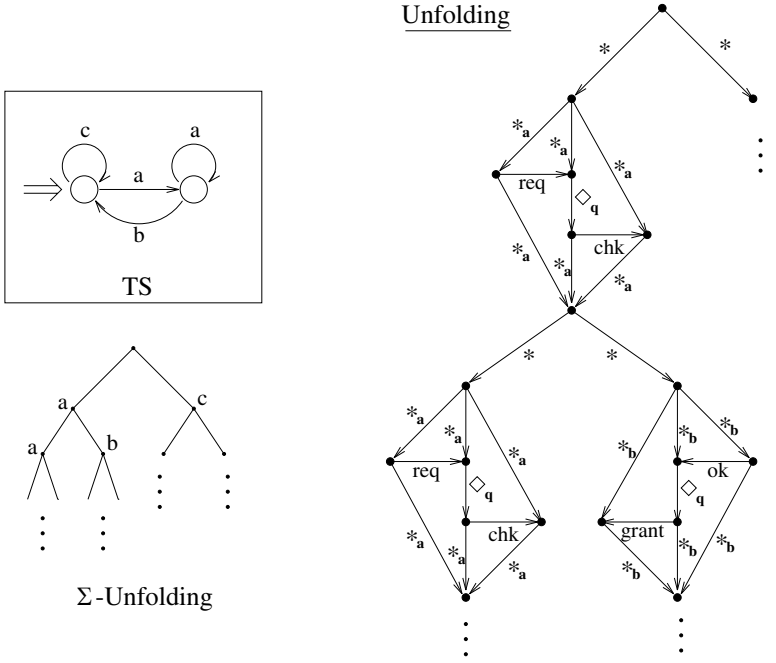


Fig. 3. A transition system, the Σ-unfolding and the unfolding

Theorem 4. *Let M be a finite set of MSCs, Σ be a finite set, $h : \Sigma \to M$ be a bijection, $D \in \mathbb{N}$ and φ be an MSO formula over Π. Then the problem of finding whether there is an MSG $\mathcal{M} = (\Sigma, TS, M, h)$ of branching degree at most D such that the unfolding of $\mathcal{M}$ satisfies φ, is decidable.*[2]

Proof: Construct, using Lemma 2, a tree automaton accepting the language $\mathcal{L}_\varphi^{\Sigma,M,h}$. Clearly, there is an unfolding formed using MSCs in M and respecting h which satisfies φ iff $\mathcal{L}_\varphi^{\Sigma,M,h} \neq \emptyset$. If the language is nonempty, we can use a regular run accepted by the automaton and construct a finite-state MSG whose unfolding satisfies φ. □

This theorem can be used to synthesize MSGs which satisfy a property φ. The new edges added in the wrap constructions allow the formula to express properties of nondeterminism which can for example arise due to the fact that the environment is uncontrollable.

Acknowledgement. I would like to thank Wolfgang Thomas, who introduced me to decidable MSO theories of graphs beyond trees, and Meenakshi, for introducing me to MSCs and helping me understand the related literature.

[2] We need not even require D to be given— however the proof of this is out of the scope of this paper.

References

1. R. Alur, G.J. Holzmann, and D. Peled. An analyzer for message sequence charts. In *Software Concepts and Tools*, volume 17(2), pages 70–77, 1996.
2. R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. CONCUR '99*, volume 1664 of *LNCS*. Springer-Verlag, 1999.
3. J. R. Büchi. Weak second-order arithmetic and finite automata. In *Z. Math. Logik Grundl. Math.*, volume 6, pages 66–92, 1960.
4. J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method and Philos. Sci. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.
5. D. Caucal. On infinite transition graphs having a decidable monadic theory. In *Proc. 23rd ICALP*, volume 1099 of *LNCS*, pages 194–205. Springer-Verlag, 1996.
6. C.C. Elgot. Decision problems of finite automata and related arithmetics. In *Trans. Amer. Math. Soc.*, volume 98, pages 21–52, 1961.
7. J.G. Henriksen, M. Mukund, Narayan Kumar, and P.S. Thiagarajan. Towards a theory of regular MSC languages. *BRICS Report RS-99-52, Department of Computer Science, Aarhus University, Denmark*, 1999.
8. J.G. Henriksen, M. Mukund, Narayan Kumar, and P.S. Thiagarajan. On message sequence graphs and finitely generated regular MSC languages. In *Proc. ICALP '00*, volume 1853 of *LNCS*. Springer-Verlag, 2000.
9. J.G. Henriksen, M. Mukund, Narayan Kumar, and P.S. Thiagarajan. Regular collections of Message Sequence Charts. In *Proc. MFCS '00*, LNCS. Springer-Verlag, 2000.
10. ITU-TS Recommendation Z.120. Message sequence chart (MSC). *ITU-TS*, 1997.
11. V. Levin and D. Peled. Verification of message sequence charts via template matching. In *Proc. TAPSOFT '97*, volume 1214 of *LNCS*, pages 652–666. Springer-Verlag, 1997.
12. P. Madhusudan. Reasoning about Sequential and Branching properties of Message Sequence Graphs Technical Report IMSC/2001/04/22, *Institute of Mathematical Sciences*, Chennai, India, 2001.
13. A. Muscholl and D. Peled. Message sequence charts and decision problems on Mazurkiewicz traces. In *Proc. MFCS '99*, volume 1672 of *LNCS*, pages 81–91. Springer-Verlag, 1999.
14. A. Muscholl, D. Peled, and Z. Su. Deciding properties of message sequence charts. In *Proc. FOSSACS '98*, volume 1378 of *LNCS*, pages 226–242. Springer-Verlag, 1998.
15. B. Meenakshi and R. Ramanujam. Reasoning about message passing in finite state environments. In *Proc. ICALP '00*, LNCS. Springer-Verlag, 2000.
16. E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on message sequence charts. In *Computer Networks and ISDN Systems—SDL and MSC, Volume 28*, 1996.
17. W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 165–191, 1990.
18. W. Thomas. Languages, automata, and logic. *Handbook of Formal Language Theory*, III:389–455, 1997.
19. P.S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for Mazurkiewicz traces. In *Proc. 12th IEEE Conf. on Logic in Computer Science*, LNCS. Springer-Verlag, 1997.
20. I. Walukiewicz. Monadic second order logic on tree-like structures. In *Proc. STACS '96*, volume 1046 of *LNCS*. Springer-Verlag, 1996.

A Set-Theoretic Framework for Assume-Guarantee Reasoning

Patrick Maier

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
maier@mpi-sb.mpg.de

Abstract. We present a circular assume-guarantee rule in an abstract setting (of sets over a partially-ordered domain). The rule has a mathematically concise side condition. Now, in order to prove an assume-guarantee rule in a concrete setting, all we need to do is to is to instantiate the abstract setting and check the side condition; i.e., we need not redo the notorious circularity argument again. We use this framework to prove a new assume-guarantee rule for Kripke structures. That rule generalizes existing assume-guarantee rules for other settings such as Reactive Modules or Mealy machines.

1 Introduction

Compositional verification tries to use the modular structure of large systems and specifications to decompose intractable verification tasks into a bunch of smaller, hopefully tractable ones. After having verified the smaller subtasks, one needs a proof rule justifying that verification of these subtasks in fact implies correctness of the complete system w.r.t. the complete specification. Essentially, there are three different kinds of such proof rules, which are applicable in different situations.

Let us consider two systems S_1 and S_2 as sets via a set-theoretic semantics, and let us assume that the composition of S_1 and S_2 amounts to the intersection $S_1 \cap S_2$. Similarly, let us view two properties P_1 and P_2 as sets and their conjunction as intersection $P_1 \cap P_2$. Our goal is to verify that $S_1 \cap S_2$ is a subset of $P_1 \cap P_2$, meaning that the system $S_1 \cap S_2$ satisfies the property $P_1 \cap P_2$.

If we are able to verify the subtasks $S_1 \subseteq P_1$ and $S_2 \subseteq P_2$, we can achieve our goal using the purely compositional proof rule (1).

$$\frac{S_1 \subseteq P_1 \qquad S_2 \subseteq P_2}{S_1 \cap S_2 \subseteq P_1 \cap P_2} \tag{1}$$

However, it may be the case that S_2 does not satisfy P_2 because S_2 assumes its environment S_1 to behave in a certain way – for instance, at some moment in time, S_2 expects some shared integer variable to have a positive value – and shows unspecified behavior if these assumptions are wrong. If the specification P_1 implies the assumptions of S_2 about the behavior of S_1, i.e., if $P_1 \cap S_2$ satisfies

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 821–834, 2001.

P_2, then we can use the hierarchical proof rule (2) to complete our verification task.

$$\frac{S_1 \subseteq P_1 \qquad P_1 \cap S_2 \subseteq P_2}{S_1 \cap S_2 \subseteq P_1 \cap P_2} \tag{2}$$

Even worse, S_1 also might have to be constrained by assumptions on its environment S_2 in order to satisfy its specification P_1. So we could end up in a situation where we can verify the subtasks $P_2 \cap S_1 \subseteq P_1$ and $P_1 \cap S_2 \subseteq P_2$, and we would need a proof rule like (3) to justify that $S_1 \cap S_2$ really satisfies $P_1 \cap P_2$.

$$\frac{P_2 \cap S_1 \subseteq P_1 \qquad P_1 \cap S_2 \subseteq P_2}{S_1 \cap S_2 \subseteq P_1 \cap P_2} \tag{3}$$

This rule states that the composition of the systems S_1 and S_2 guarantees the conjunction of the properties P_1 and P_2, provided that S_1 guarantees P_1 if P_2 is assumed, whereas S_2 guarantees P_2 if P_1 is assumed. Thus in both subtasks, the guaranteed property relies on the other one as an assumption, hence such rules are named *circular assume-guarantee rules*.

While rules like (1) and (2) simply follow from monotonicity of intersection and therefore are valid in every context where there is a set-theoretic semantics that treats composition as intersection, this is not the case for circular rules. Set-theoretically, rule (3) is only valid if we weaken its conclusion to $S_1 \cap S_2 \subseteq P_1 \cap P_2 \cup \bar{P}_1 \cap \bar{P}_2$ where $\bar{P}_1$ and $\bar{P}_2$ are the complements of P_1 and P_2, respectively.

However, circular assume-guarantee rules in the style of (3) have been proposed for various modeling formalisms, for instance Reactive Modules [3]. These formalisms augment the unstructured set-theoretic semantics of systems and properties with additional structure, thus providing ways to break the circularity in the assumptions by an inductive argument. The induction itself relies on a special 'small-step semantics', for example the *partial traces* of Reactive Modules, which is usually introduced solely for the purpose of proving the assume-guarantee rule.

We pursue a different approach. Starting from a particularly simple modeling formalism – downward-closed subsets of a poset – we prove a generic circular assume-guarantee rule that is valid if a concise side condition is true. This rule is useful as a pattern from which rules in more complex formalisms can be derived. To derive a new rule in a particular formalism, one just has to instantiate the generic rule to the semantics of the target formalism and prove that the side condition holds. Thus, one need not prove a circularity-breaking induction any more, as the circularity of the assumptions has been factored out into the generic rule.

Besides simplifying the derivation of new circular rules, there are at least two reasons in favor of our approach. First, the proofs become a lot clearer. For proving the generic assume-guarantee rule, we work with a very simple formalism. More complicated semantics only play a role when instantiating the framework to other formalisms, which amounts to proving that the target formalism meets a small number of well-expressed requirements. Thus, there is a clear separation between the proof rule as a general principle and the verifier's modeling formalism as a means to express this principle in a form which makes it applicable to real verification tasks.

Second, our framework provides some insight into what conditions must hold generally of modeling formalisms in order to allow circular assume-guarantee reasoning and what restrictions could safely be dropped. This should help in designing modeling formalisms that deliberately support circular assume-guarantee reasoning.

2 The Framework

In this section, we work in the abstract setting of subsets of a partially ordered universe and prove our generic assume-guarantee rule for set inclusion.

Behaviors and Chains. We fix a partially ordered universe $(\mathcal{B}, \preceq)$ with bottom ϵ. Elements $b, b' \in \mathcal{B}$ are called *behaviors*, and $b \preceq b'$ intuitively means that the behavior b can evolve into b'. A set $B \subseteq \mathcal{B}$ is *downward-closed* if for all $b \in B$ and $b' \in \mathcal{B}$, $b' \preceq b$ implies $b' \in B$. A set $B' \subseteq \mathcal{B}$ is the *downward-closure* of $B \subseteq \mathcal{B}$ if B' is the least downward-closed superset of B.

Specifically to describe the evolution of behaviors, we introduce ascending sequences of behaviors that continuously reveal more information. We call a sequence $(b_i)_{i\in\mathbb{N}}$ of behaviors in $\mathcal{B}$ a *chain* if it ascends from the bottom and converges to some limit in $\mathcal{B}$. More precisely, $(b_i)_{i\in\mathbb{N}}$ is a chain, if $b_0 = \epsilon$, $b_i \preceq b_{i+1}$ for all $i \in \mathbb{N}$, and the least upper bound $b \in \mathcal{B}$ of the set $\{b_i \mid i \in \mathbb{N}\}$ exists.[1] This unique behavior b is called the *limit* of the chain $(b_i)_{i\in\mathbb{N}}$, and it is denoted by $lim(b_i)_{i\in\mathbb{N}}$.

Extension. Given a chain $(b_i)_{i\in\mathbb{N}}$ and two sets $B_1, B_2 \subseteq \mathcal{B}$, we say that B_1 and B_2 *extend* along $(b_i)_{i\in\mathbb{N}}$ if b_{i+1} is contained in at least one of them whenever b_i has been contained in both, i.e., if for all $i \in \mathbb{N}$, $b_i \in B_1 \cap B_2$ implies $b_{i+1} \in B_1 \cup B_2$. For instance, the sets $B_1 = \{\epsilon, b, b'\}$ and $B_2 = \{\epsilon, b\}$ extend along the chain $(b_i)_{i\in\mathbb{N}}$ with $b_0 = \epsilon$, $b_1 = b$, $b_2 = b_3 = \ldots = b'$.

Closedness. Given a chain $(b_i)_{i\in\mathbb{N}}$ and a set $B \subseteq \mathcal{B}$, we say that B is *closed* w.r.t. $(b_i)_{i\in\mathbb{N}}$ if the limit of $(b_i)_{i\in\mathbb{N}}$ lies in B whenever each element b_i lies in B, i.e., if $lim(b_i)_{i\in\mathbb{N}} \in B$ is implied by $\forall i \in \mathbb{N}\ b_i \in B$. For example, B is closed w.r.t. $(b_i)_{i\in\mathbb{N}}$ if the set $\{b_i \mid i \in \mathbb{N}\}$ is finite.

Assume-Guarantee Rule. The stage is set for our main theorem, which justifies the circular assume-guarantee proof rule for non-empty downward-closed subsets of $\mathcal{B}$. Non-emptiness just rules out pathological cases. If one thinks of these sets as properties (named P_1, P_2) and systems (named S_1, S_2) then downward-closure is a natural requirement, too. A system exhibiting a particular behavior b should also exhibit all the 'earlier' behaviors $b' \preceq b$.

[1] We do not call $(b_i)_{i\in\mathbb{N}}$ a chain, if in $\mathcal{B}$ the least upper bound of $\{b_i \mid i \in \mathbb{N}\}$ does not exist.

Theorem 1 *Let P_1, P_2, S_1, S_2 be non-empty downward-closed subsets of a partially ordered universe $(\mathcal{B}, \preceq)$ with bottom ϵ. Then the rule*

$$\frac{P_2 \cap S_1 \subseteq P_1 \qquad P_1 \cap S_2 \subseteq P_2}{S_1 \cap S_2 \subseteq P_1 \cap P_2} \tag{4}$$

holds if for every $b \in \mathcal{B}$ there exists a chain $(b_i)_{i\in\mathbb{N}}$ converging to b, such that

1. *P_1 and P_2 extend along $(b_i)_{i\in\mathbb{N}}$, and*
2. *P_1 and P_2 both are closed w.r.t. $(b_i)_{i\in\mathbb{N}}$.*

Proof. To prove rule (4), we assume that its premises hold and choose $b \in S_1 \cap S_2$ arbitrary. Then there is a chain $(b_i)_{i\in\mathbb{N}}$ converging to b such that conditions 1 and 2 hold. By induction on i, we show that $b_i \in P_1 \cap P_2$ for all $i \in \mathbb{N}$.

- $i = 0$. Then by definition of chains $b_i = \epsilon$. As P_1 and P_2 are non-empty and downward-closed, the bottom ϵ is contained in both, hence $b_i \in P_1 \cap P_2$.
- $i > 0$. Then by induction hypothesis, $b_{i-1} \in P_1 \cap P_2$, hence by extension (condition 1), $b_i \in P_1 \cup P_2$. We have to distinguish two cases.
  First, assume $b_i \in P_1$. The limit of $(b_i)_{i\in\mathbb{N}}$, b, is contained in S_2 by choice, and as S_2 is downward-closed, $b_i \in S_2$ as well. Therefore, $b_i \in P_1 \cap S_2$, and by the second premise of rule (4), $b_i \in P_2$. Hence, $b_i \in P_1 \cap P_2$.
  Second, assume $b_i \in P_2$. The proof that this implies $b_i \in P_1 \cap P_2$ is a completely symmetric variant of the first case.

This ends the inductive proof that $b_i \in P_1 \cap P_2$ for all $i \in \mathbb{N}$. By closedness (condition 2), the limit b of $(b_i)_{i\in\mathbb{N}}$ is contained in both, P_1 and P_2, hence $b \in P_1 \cap P_2$. As $b \in S_1 \cap S_2$ has been chosen arbitrary, the conclusion of rule (4) holds. □

Two remarks on this theorem are due. First, the conclusion of rule (4) is vacuously true if $S_1 \cap S_2 = \emptyset$, so non-emptiness of S_1 and S_2 is not strictly required. Second, the side condition could be weakened to 'for every $b \in S_1 \cap S_2$' instead of 'for every $b \in \mathcal{B}$'. However, that would contradict the spirit of compositional reasoning, which seeks to avoid the unmanageable system composition $S_1 \cap S_2$ in premises and side conditions.

Instantiations. To instantiate the framework to a particular modeling formalism, one must provide an ordered universe with bottom, and semantics that map systems and properties to non-empty downward-closed sets. Then one has to prove that these semantics entail the side condition of Theorem 1. Typically, this requires to restrict to certain pairs of properties, i.e., one must find a criterion that filters pairs of properties which extend along sufficiently many chains and are closed w.r.t. these chains. Usually, closedness is not the problem, as many properties (e.g., all safety properties) are trivially closed w.r.t. every chain. Thus, if the criterion holds for a pair of properties, it has to ensure that every behavior is the limit of some chain along which this pair extends. Following this recipe, one can derive a rule, whose side condition is a criterion ensuring extension and which is expressed in terms of the semantics of the modeling formalism.

3 Instantiation to Moore Machines

To exemplify how to instantiate our framework to a concrete setting, we examine the setting where both, systems and properties are presented in a simple synchronous formalism, namely Moore machines (cf. [5]). Thus, we derive a well-known assume-guarantee rule for Moore machines under trace semantics.

Notation. We fix a finite non-empty set of variables $\mathcal{X}$ and a (possibly infinite) non-empty domain of values $\mathcal{D}$. By $a = \left(\begin{smallmatrix} x_1 & \cdots & x_n \\ d_1 & \cdots & d_n \end{smallmatrix}\right)$ we denote the partial map from $\mathcal{X}$ to $\mathcal{D}$ whose domain $dom(a)$ is $\{x_1, \ldots, x_n\}$ and which maps every x_i to d_i. We call a a partial label. The set of partial labels Σ is ordered by $\preceq$, where $a \preceq a'$ if $dom(a) \subseteq dom(a')$ and $a(x) = a'(x)$ for all $x \in dom(a)$. The undefined map $\bot$ is the bottom of Σ. The maximal elements of Σ, the total maps, form the set of labels Σ_t.

Given $a_1, a_2 \in \Sigma$ with common upper bound in Σ, their sum $a_1 \oplus a_2$ is their least upper bound in Σ. Given $a \in \Sigma$ and $X \subseteq \mathcal{X}$, the restriction $a|_X$ is the partial label $a' \preceq a$ with $dom(a') = dom(a) \cap X$.

We write a finite word $w \in \Sigma^*$ as a finite sequence $w = a_1 \ldots a_n$, $a_i \in \Sigma$; we write $w = \epsilon$ for the empty word. The set Σ^* is ordered by the lexicographic extension of $\preceq$, which we also denote by $\preceq$.

Moore Machines. A *Moore machine* M is a six-tuple $(I, O, S, S^\iota, \delta, \rho)$, where $I \subseteq \mathcal{X}$ and $O \subseteq \mathcal{X}$ are disjoint sets of input resp. output variables, S is the (possibly infinite) state space, $S^\iota \subseteq S$ are the initial states, $\delta : S \times \Sigma_\mathrm{t}|_I \to 2^S$ is the transition function and $\rho : S \to \Sigma_\mathrm{t}|_O$ the output function. M must be *non-blocking*, i.e., $S^\iota \neq \emptyset$ and $\delta(s, a) \neq \emptyset$ for all $s \in S$ and all $a \in \Sigma_\mathrm{t}|_I$.

Let $M_1 = (I_1, O_1, S_1, S_1^\iota, \delta_1, \rho_1)$ and $M_2 = (I_2, O_2, S_2, S_2^\iota, \delta_2, \rho_2)$ be Moore machines. We call M_1 and M_2 *compatible* if their sets of output variables O_1 and O_2 are disjoint. In this case, we can form the *(synchronous) product* $M_1 \parallel M_2$ of M_1 and M_2 by 'connecting' inputs of M_1 to outputs of M_2 and vice versa; formally $M_1 \parallel M_2 = \big((I_1 \cup I_2) \setminus (O_1 \cup O_2), O_1 \cup O_2, S_1 \times S_2, S_1^\iota \times S_2^\iota, \delta, \rho\big)$, where

$$\delta\big((s_1, s_2), a\big) = \delta_1\big(s_1, (a \oplus \rho_2(s_2))|_{I_1}\big) \times \delta_2\big(s_2, (a \oplus \rho_1(s_1))|_{I_2}\big) \text{ and}$$
$$\rho\big((s_1, s_2)\big) = \rho_1(s_1) \oplus \rho_2(s_2).$$

Clearly, $M_1 \parallel M_2$ is non-blocking, hence a Moore machine. For the remainder of this section, we assume the set of variables $\mathcal{X}$ to be $\{x, y\}$ and consider only products $M_1 \parallel M_2$ where $I_1 = \{x\} = O_2$ and $I_2 = \{y\} = O_1$; these restrictions avoid some technicalities, yet all results continue to hold for the general case.

Trace Semantics. Let $M = (I, O, S, S^\iota, \delta, \rho)$ be a Moore machine. We call a word $w = a_1 \ldots a_n \in \Sigma_\mathrm{t}^*$ a *trace* of M if there exists a *run* of w in M, i.e., if there exists a sequence $(s_1, \ldots, s_n)$ of states in S such that $s_1 \in S^\iota$, $s_{i+1} \in \delta(s_i, a_i|_I)$ for all $i < n$, and $a_i|_O = \rho(s_i)$ for all $i \leq n$; note that the null sequence $()$ is a run of ϵ in M. The *trace language* $[\![M]\!]$ is the set of all traces of M. It is easy to see that the trace semantics maps products to intersections, i.e., $[\![M_1 \parallel M_2]\!] = [\![M_1]\!] \cap [\![M_2]\!]$ for compatible Moore machines M_1 and M_2.

To instantiate our framework to Moore machines, we could now choose $(\Sigma_{\mathrm{t}}^*, \preceq)$ as our ordered universe of behaviors. Then the trace semantics maps Moore machines to non-empty downward-closed subsets of Σ_{t}^*. Besides, every subset of Σ_{t}^* is trivially closed w.r.t. every chain, since $\{w_i \mid i \in \mathbb{N}\}$ is finite for every chain $(w_i)_{i\in\mathbb{N}}$ in Σ_{t}^*. Unfortunately, we cannot instantiate the framework in such a straightforward way, as the following example shows that extension does not hold.

Example 1 Let M_1 and M_2 be 'crosswired' latches, i.e., the input of M_1 is the output of M_2 and vice versa. Formally, $M_1 = \big(\{x\}, \{y\}, \mathcal{D}, \mathcal{D}, \delta_1, \rho_1\big)$ is a Moore machine, where $\delta_1\big(s, \left(\begin{smallmatrix} x \\ d \end{smallmatrix}\right)\big) = \{d\}$ and $\rho_1(s) = \left(\begin{smallmatrix} y \\ s \end{smallmatrix}\right)$, and M_2 is a copy of M_1 with x and y swapped. All traces of M_1 are of the form $\left(\begin{smallmatrix} x & y \\ d_1 & d_0 \end{smallmatrix}\right)\left(\begin{smallmatrix} x & y \\ d_2 & d_1 \end{smallmatrix}\right)\cdots\left(\begin{smallmatrix} x & y \\ d_n & d_{n-1} \end{smallmatrix}\right)$, whereas all traces of M_2 are of the form $\left(\begin{smallmatrix} x & y \\ d_0 & d_1 \end{smallmatrix}\right)\left(\begin{smallmatrix} x & y \\ d_1 & d_2 \end{smallmatrix}\right)\cdots\left(\begin{smallmatrix} x & y \\ d_{n-1} & d_n \end{smallmatrix}\right)$. As a counterexample for extension, take $w = \left(\begin{smallmatrix} x & y \\ d_0 & d_0 \end{smallmatrix}\right)\left(\begin{smallmatrix} x & y \\ d_1 & d_1 \end{smallmatrix}\right) \in \Sigma_{\mathrm{t}}^*$ for some $d_0 \neq d_1$, and let $(w_i)_{i\in\mathbb{N}}$ be a chain converging to w. There is a minimal $i \in \mathbb{N}$ such that $w_{i+1} = w$, so either $w_i = \epsilon$ or $w_i = \left(\begin{smallmatrix} x & y \\ d_0 & d_0 \end{smallmatrix}\right)$. In any case, $w_i \in [\![M_1]\!] \cap [\![M_2]\!]$ but $w_{i+1} \notin [\![M_1]\!] \cup [\![M_2]\!]$, hence $[\![M_1]\!]$ and $[\![M_2]\!]$ do not extend along $(w_i)_{i\in\mathbb{N}}$.

To summarize, the trace semantics is too coarse to instantiate our assume-guarantee framework, because it can force a common trace of M_1 and M_2 – here $\left(\begin{smallmatrix} x & y \\ d_0 & d_0 \end{smallmatrix}\right)$ – to evolve in a single indivisible step into a trace of neither – here $\left(\begin{smallmatrix} x & y \\ d_0 & d_0 \end{smallmatrix}\right)\left(\begin{smallmatrix} x & y \\ d_1 & d_1 \end{smallmatrix}\right)$. Therefore, we need a more fine-grained semantics with smaller indivisible steps such that no joint trace of M_1 and M_2 can evolve into a trace of neither in a single indivisible step.

Partial Trace Semantics. Let $\Sigma_{\mathrm{suff}} = \{a \in \Sigma \mid x \in dom(a)\}$ be those partial labels whose domain is either $\{x\}$ or $\{x, y\}$. We define our universe of behaviors $\mathcal{B}$ as the set of *partial words* $\{\epsilon\} \cup \Sigma_{\mathrm{t}}^* \Sigma_{\mathrm{suff}}$, which is ordered by $\preceq$.

For every chain $(w_i)_{i\in\mathbb{N}}$, the set $\{w_i \mid i \in \mathbb{N}\}$ is contained in the downward-closure of $\{lim(w_i)_{i\in\mathbb{N}}\}$, which is finite. Therefore, closedness is trivial in $(\mathcal{B}, \preceq)$.

Proposition 1 *Let $B \subseteq \mathcal{B}$, and let $(w_i)_{i\in\mathbb{N}}$ be a chain. Then B is closed w.r.t. $(w_i)_{i\in\mathbb{N}}$.*

The set of *partial traces* $(\!|M|\!)$ of a Moore machine M is the downward-closure of $[\![M]\!]$ in $(\mathcal{B}, \preceq)$. The partial trace semantics $(\!|\cdot|\!)$ is similar to the trace semantics $[\![\cdot]\!]$ as far as inclusion and product are concerned.[2]

Proposition 2 *Let M_1, M_2 be Moore machines. Then $[\![M_1]\!] \subseteq [\![M_2]\!]$ iff $(\!|M_1|\!) \subseteq (\!|M_2|\!)$.*

Proposition 3 $(\!|M_1 \parallel M_2|\!) = (\!|M_1|\!) \cap (\!|M_2|\!)$ *for compatible Moore machines M_1, M_2.*

[2] Proofs which are omitted here due to space limitations, can be found in [8].

Extension. The main difference between the trace semantics $[\![\cdot]\!]$ and the partial trace semantics $(\!|\cdot|\!)$ is that the former does not support extension whereas the latter does. We say that $w' \in \mathcal{B}$ is an *immediate successor* of $w \in \mathcal{B}$ if $w \precneqq w'$ and there is no $w'' \in \mathcal{B}$ such that $w \precneqq w'' \precneqq w'$. Given two compatible Moore machines M_1 and M_2, we can prove that immediate successors satisfy the one-step extension criterion, i.e., if $w' \in \mathcal{B}$ is an immediate successor of $w \in \mathcal{B}$ then $w \in (\!|M_1|\!) \cap (\!|M_2|\!)$ implies $w' \in (\!|M_1|\!) \cup (\!|M_2|\!)$. Now, for every $w \in \mathcal{B}$, we can construct a chain $(w_i)_{i\in\mathbb{N}}$ such that the set $\{w_i \mid i \in \mathbb{N}\}$ equals the downward-closure of $\{w\}$, hence w_{i+1} is an immediate successor of w_i for all $i \in \mathbb{N}$. Therefore, $(w_i)_{i\in\mathbb{N}}$ is a chain along which $(\!|M_1|\!)$ and $(\!|M_2|\!)$ do extend.

Proposition 4 *Let M_1, M_2 be compatible Moore machines, and let $w \in \mathcal{B}$. Then there exists a chain $(w_i)_{i\in\mathbb{N}}$ with limit w such that $(\!|M_1|\!)$ and $(\!|M_2|\!)$ extend along $(w_i)_{i\in\mathbb{N}}$.*

Example 1 (contd.) To illustrate how the partial trace semantics supports extension, again look at $w = \left(\begin{smallmatrix} x & y \\ d_0 & d_0 \end{smallmatrix}\right)\left(\begin{smallmatrix} x & y \\ d_1 & d_1 \end{smallmatrix}\right)$ for $d_0 \neq d_1$. Take the chain $(w_i)_{i\in\mathbb{N}}$ with $w_0 = \epsilon$, $w_1 = \left(\begin{smallmatrix} x \\ d_0 \end{smallmatrix}\right)$, $w_2 = \left(\begin{smallmatrix} x & y \\ d_0 & d_0 \end{smallmatrix}\right)$, $w_3 = \left(\begin{smallmatrix} x & y \\ d_0 & d_0 \end{smallmatrix}\right)\left(\begin{smallmatrix} x \\ d_1 \end{smallmatrix}\right)$, $w_4 = w_5 = \ldots = \left(\begin{smallmatrix} x & y \\ d_0 & d_0 \end{smallmatrix}\right)\left(\begin{smallmatrix} x & y \\ d_1 & d_1 \end{smallmatrix}\right)$. For $i < 3$, w_i is a partial trace of both M_1 and M_2, for $i = 3$, w_i is a partial trace of M_1 only, and for $i > 3$, w_i is partial trace of neither. Hence for all $i \in \mathbb{N}$, $w_i \in (\!|M_1|\!) \cap (\!|M_2|\!)$ implies $w_{i+1} \in (\!|M_1|\!) \cup (\!|M_2|\!)$.

Assume-Guarantee Rule. Now, we have an ordered universe $(\mathcal{B}, \preceq)$ with bottom ϵ, and a semantics $(\!|\cdot|\!)$ that maps systems and properties (both given as Moore machines) to non-empty downward-closed sets. Furthermore, the above propositions on extension and closedness entail the side condition of Theorem 1. This lets us derive an assume-guarantee rule for the partial trace semantics of Moore machines.

Theorem 2 *Let P_1, P_2, S_1, S_2 be Moore machines such that P_1 and P_2 are compatible. Then rule (5) is sound.*

$$\frac{(\!|P_2|\!) \cap (\!|S_1|\!) \subseteq (\!|P_1|\!) \qquad (\!|P_1|\!) \cap (\!|S_2|\!) \subseteq (\!|P_2|\!)}{(\!|S_1|\!) \cap (\!|S_2|\!) \subseteq (\!|P_1|\!) \cap (\!|P_2|\!)} \tag{5}$$

Proof. Assume that P_1 and P_2 are compatible. To prove (5) to be an instance of our generic rule (4) from Theorem 1, we have to check that theorem's side condition. Indeed that condition is met, as for every $w \in \mathcal{B}$, by Proposition 4 there exists a chain $(w_i)_{i\in\mathbb{N}}$ converging to w such that

1. $(\!|P_1|\!)$ and $(\!|P_2|\!)$ extend along $(w_i)_{i\in\mathbb{N}}$, and
2. by Proposition 1, both, $(\!|P_1|\!)$ and $(\!|P_2|\!)$ are closed w.r.t. this chain $(w_i)_{i\in\mathbb{N}}$. □

The above assume-guarantee rule is hardly applicable for verification, for usually, systems and properties are not specified w.r.t. the partial trace semantics $(\!|\cdot|\!)$ but w.r.t. the coarser trace semantics $[\![\cdot]\!]$. Fortunately, the two semantics are related closely, so that as a corollary, we can derive an assume-guarantee rule for the trace semantics $[\![\cdot]\!]$.

Corollary 1 *Let P_1, P_2, S_1, S_2 be Moore machines such that P_1 and P_2, P_1 and S_2, S_1 and P_2 as well as S_1 and S_2 all are compatible. Then rule (6) is sound.*

$$\frac{[\![P_2 \parallel S_1]\!] \subseteq [\![P_1]\!] \qquad [\![P_1 \parallel S_2]\!] \subseteq [\![P_2]\!]}{[\![S_1 \parallel S_2]\!] \subseteq [\![P_1 \parallel P_2]\!]} \tag{6}$$

Proof. We assume all required compatibilities, and we assume the premises of rule (6) to hold. By Propositions 2 and 3, the first premise is equivalent to $(\!|P_2|\!) \cap (\!|S_1|\!) \subseteq (\!|P_1|\!)$. Likewise, the second premise is equivalent to $(\!|P_1|\!) \cap (\!|S_2|\!) \subseteq (\!|P_2|\!)$. So by Theorem 2, we infer $(\!|S_1|\!) \cap (\!|S_2|\!) \subseteq (\!|P_1|\!) \cap (\!|P_2|\!)$. Again by Propositions 3 (twice) and 2, this is equivalent to the conclusion of rule (6). □

4 Instantiation to Kripke Structures

Kripke structures equipped with a synchronous product are a powerful and mathematically elegant formalism for modeling synchronous computation. Besides, they provide a straightforward method to deal with shared memory in a synchronous formalism, which we are going to demonstrate by means of an example. We instantiate our framework to the setting where both, systems and properties are presented as first-order Kripke structures; thus, we derive a new circular assume-guarantee rule for Kripke structures under trace semantics. The instantiation proceeds essentially along the same lines than the one for Moore machines; recall the notation of the previous section.

Kripke Structures. A *(Kripke) structure* $K = (S, \iota, \delta, \rho)$ is a vertex-labeled directed graph with vertices S, source vertex ι, edges δ and labeling ρ. More precisely, the *state space* S is a (possibly infinite) set containing the *initial state* ι. The *transition function* $\delta : S \to 2^S$ maps states to sets of successor states, and the *labeling function* $\rho : S \to \Sigma_t$ maps states to labels. By convention, all Kripke structures label their initial state with the same label $\rho(\iota)$.[3] We call K *non-blocking* if $\delta(s) \neq \emptyset$ for every state s which is reachable in K, where s is *reachable* in K if $s \in \delta^n(\{\iota\})$ for some $n \in \mathbb{N}$.

Let $K_1 = (S_1, \iota_1, \delta_1, \rho_1)$ and $K_2 = (S_2, \iota_1, \delta_2, \rho_2)$ be structures. Their *(synchronous) product* $K_1 \parallel K_2$ is defined as (S, ι, δ, ρ), where

$$\begin{aligned} S &= \{(s_1, s_2) \in S_1 \times S_2 \mid \rho_1(s_1) = \rho_2(s_2)\}, \\ \iota &= (\iota_1, \iota_2), \\ \delta\big((s_1, s_2)\big) &= \{(s_1', s_2') \in S \mid s_1' \in \delta_1(s_1), s_2' \in \delta_2(s_2)\}, \text{ and} \\ \rho\big((s_1, s_2)\big) &= \rho_1(s_1) = \rho_2(s_2). \end{aligned}$$

Clearly, $K_1 \parallel K_2$ is a Kripke structure. Note that the product $K_1 \parallel K_2$ of any two Kripke structures K_1 and K_2 is defined, unlike the product of Moore machines. However, $K_1 \parallel K_2$ need not be non-blocking, not even if both components are.

[3] Semantically, the labeling of the initial state is irrelevant; this convention just facilitates the definition of the product.

Fig. 1. Asynchronous producer-consumer protocol; $q \in \mathbb{N}^*$ denotes the content of the shared FIFO-buffer before, q' after a transition

Producer $P = (S_P, \iota_P, \delta_P, \rho_P)$ where Consumer $C = (S_C, \iota_C, \delta_C, \rho_C)$ where

$$S_P = S_C = \Sigma_{\mathrm{t}},$$
$$\iota_P = \iota_C = \left(\begin{smallmatrix} p & q \\ 0 & \epsilon \end{smallmatrix}\right),$$
$$\rho_P(s) = \rho_C(s) = s,$$
$$\delta_P(s) = \{s' \mid s'(p) \neq 0 \vee \big(s'(p) = 0 \wedge (s'(q) = s(q) \vee \exists i \in \mathbb{N}\ s'(q) = s(q).i)\big)\},$$
$$\delta_C(s) = \{s' \mid s'(p) \neq 1 \vee \big(s'(p) = 1 \wedge (s'(q) = s(q) \vee \exists i \in \mathbb{N}\ s(q) = i.s'(q))\big)\}.$$

Fig. 2. Synchronous implementation of the producer-consumer protocol using Kripke structures, where variables $\mathcal{X} = \{p, q\}$, domain $\mathcal{D} = \mathcal{D}_p \cup \mathcal{D}_q$ with $\mathcal{D}_p = \{0, 1\}$ and $\mathcal{D}_q = \mathbb{N}^*$, and alphabet $\Sigma_{\mathrm{t}} = \{a : \mathcal{X} \to \mathcal{D} \mid a(p) \in \mathcal{D}_p, a(q) \in \mathcal{D}_q\}$

Example 2 To illustrate Kripke structures, we encode the simple producer-consumer protocol depicted in figure 1. The processes communicate over a shared FIFO-buffer q, to which the producer may enqueue a number or stutter, whereas the consumer may dequeue a number if q is non-empty, or stutter. This protocol works perfectly in an asynchronous setting, which prevents simultaneous updates of the shared buffer q.

In a synchronous setting, Kripke structures admit write-shared variables, i.e., variables may be updated by several processes — but not simultaneously. Therefore, we can directly encode asynchronous protocols like the above, if we can prevent simultaneous updates of shared variables. For that we introduce a new shared variable p reflecting whether the producer P or the consumer C is the currently active process. If the value of p becomes 0 (1) then P (C) executes and may update q while C (P) does not access q. Figure 2 presents the structures P and C in detail. Note that there is no such straightforward encoding of the above protocol into Moore or Mealy machines as these formalisms do not not admit write-shared variables.

Trace Semantics. Let $K = (S, \iota, \delta, \rho)$ be a structure. A word $w = a_1 \dots a_n \in \Sigma_{\mathrm{t}}^*$ is a *trace* of K if there exists a *run* of w in M, i.e., if there exists a sequence $(s_1, \dots, s_n)$ of states in S such that $s_1 \in \delta(\iota)$, $s_{i+1} \in \delta(s_i)$ for all $i < n$, and $a_i = \rho(s_i)$ for all $i \leq n$. The *trace language* $[\![K]\!]$ is the set of all traces of K. Note that $[\![K]\!] \neq \emptyset$ because the null sequence $()$ is a run of ϵ in K. As expected, the trace semantics maps products to intersections, i.e., $[\![K_1 \parallel K_2]\!] = [\![K_1]\!] \cap [\![K_2]\!]$ for all structures K_1 and K_2.

Partial Trace Semantics. As is the case for Moore machines, the trace semantics $[\![\cdot]\!]$ is too coarse to instantiate our assume-guarantee framework. We need a more fine-grained partial trace semantics $(\!|\cdot|\!)$ whose indivisible steps are

small enough to support extension. Its definition is essentially the same as in the previous section, only the construction of Σ_{suff} is more involved.

We fix a function $next : \Sigma \setminus \Sigma_{\text{t}} \to \mathcal{X}$ that maps (truly) partial labels to variables such that $next(a) \notin dom(a)$. This function induces a set Σ_{suff} of partial labels – quasi the closure of the undefined label $\bot$ under $next(\cdot)$ – which is defined as the least set containing $\bot$ and being closed under the rule that for all $a \in \Sigma$, if $a \in \Sigma_{\text{suff}} \setminus \Sigma_{\text{t}}$ then $\big\{a' \in \Sigma \mid a \preceq a',\ dom(a') = dom(a) \cup \{next(a)\}\big\} \subseteq \Sigma_{\text{suff}}$. We define our universe of behaviors $\mathcal{B}$ as the set of *partial words* $\{\epsilon\} \cup \Sigma_{\text{t}}^* \Sigma_{\text{suff}}$, ordered by $\preceq$. Like in the previous section, closedness is trivial in $(\mathcal{B}, \preceq)$.

Proposition 5 *Let $B \subseteq \mathcal{B}$, and let $(w_i)_{i\in\mathbb{N}}$ be a chain. Then B is closed w.r.t. $(w_i)_{i\in\mathbb{N}}$.*

The set of *partial traces* $(\!|K|\!)$ of a structure K is the downward-closure of $[\![K]\!]$ in $(\mathcal{B}, \preceq)$. Note that implicitly, $(\!|K|\!)$ depends on our choice for the $next(\cdot)$ function. Like before, the trace semantics $[\![\cdot]\!]$ and the partial trace semantics $(\!|\cdot|\!)$ are equivalent w.r.t. inclusion.

Proposition 6 *Let K_1 and K_2 be structures. Then $[\![K_1]\!] \subseteq [\![K_2]\!]$ iff $(\!|K_1|\!) \subseteq (\!|K_2|\!)$.*

Labeling Games. For Moore machines, the partial trace semantics maps a product to an intersection, whenever this product is defined. For Kripke structures K_1 and K_2, the product is always defined, yet $(\!|K_1 \parallel K_2|\!)$ may be truly contained in $(\!|K_1|\!) \cap (\!|K_2|\!)$. We present a game which illuminates the origin of the $next(\cdot)$ function and leads to a sufficient criterion ensuring that the partial trace semantics maps products to intersections.

Let $K_1 \parallel K_2 = (S, \iota, \delta, \rho)$ be the product of two structures K_1 and K_2. Given a state $(s_1, s_2) \in S$, we want to determine a labeling $\rho\big((s_1', s_2')\big)$ of some successor state $(s_1', s_2') \in \delta\big((s_1, s_2)\big)$ by playing a challenge-response game over n rounds, where n is the cardinality of $\mathcal{X}$. In round i, the challenger C picks a yet undefined variable $x_i \in \mathcal{X}$, to which the responder R assigns a value $d_i \in \mathcal{D}$. In the end, $\left(\begin{smallmatrix} x_1 & \dots & x_n \\ d_1 & \dots & d_n \end{smallmatrix}\right) \in \Sigma_{\text{t}}$ is a label, and R wins this labeling game if $\delta\big((s_1, s_2)\big) = \emptyset$ or if there exists some $(s_1', s_2') \in \delta\big((s_1, s_2)\big)$ with $\rho\big((s_1', s_2')\big) = \left(\begin{smallmatrix} x_1 & \dots & x_n \\ d_1 & \dots & d_n \end{smallmatrix}\right)$, otherwise C wins. Obviously, strategies for C correspond to $next(\cdot)$ functions and vice versa, so the $next(\cdot)$ function which determines our universe $\mathcal{B}$, also induces a one-player labeling game for R.

Given a structure $K = (S, \iota, \delta, \rho)$ and a partial label $a \in \Sigma$, we define the *a-filtered* transition function $\delta^a : S \to 2^S$ by $\delta^a(s) = \{s' \in \delta(s) \mid a \preceq \rho(s')\}$, i.e., δ^a filters transitions to states whose labeling is consistent with a. We say that $a' \in \Sigma$ is an *immediate successor* of $a \in \Sigma$ if $a \not\preceq a'$ and there is no $a'' \in \Sigma$ such that $a \not\preceq a'' \not\preceq a'$. Two structures $K_1 = (S_1, \iota_1, \delta_1, \rho_1)$ and $K_2 = (S_2, \iota_2, \delta_2, \rho_2)$ are *compatible (w.r.t. $next(\cdot)$)* if for every reachable state (s_1, s_2) in $K_1 \parallel K_2$ and every $a \in \Sigma_{\text{suff}} \setminus \Sigma_{\text{t}}$,

- $\delta_1^a(s_1) \neq \emptyset$ implies that $\delta_1^{a'}(s_1) \neq \emptyset$ for all immediate successors a' of a, or
- $\delta_2^a(s_2) \neq \emptyset$ implies that $\delta_2^{a'}(s_2) \neq \emptyset$ for all immediate successors a' of a.

In this case, for every reachable state (s_1, s_2) in $K_1 \parallel K_2$, there exists a winning strategy for R in the one-player labeling game. The existence of such a strategy allows to prove that the product of compatible non-blocking structures is non-blocking, and moreover, that the partial trace semantics maps products of compatible structures to intersections.

Proposition 7 *Let K_1 and K_2 be structures. Then $(\!|K_1 \parallel K_2|\!) \subseteq (\!|K_1|\!) \cap (\!|K_2|\!)$, and $(\!|K_1 \parallel K_2|\!) = (\!|K_1|\!) \cap (\!|K_2|\!)$ if K_1 and K_2 are compatible.*

Example 2 (contd.) To show that compatible structures exist, we check that P and C are compatible. First note that the state space of $P \parallel C$ equals $\{(s, s) \mid s \in \Sigma_{\mathrm{t}}\}$, and that this full state space is reachable in $P \parallel C$. Choose the $next(\cdot)$ function that maps $\bot$ to p. Then the compatibility condition holds for all states (s, s) of $P \| C$ and all partial labels $a \in \Sigma_{\mathrm{suff}} \backslash \Sigma_{\mathrm{t}}$, because if $a = \bot$ or $a = \binom{p}{1}$ then $\delta_P^{a'}(s) \neq \emptyset$ for all immediate successors a' of a, and if $a = \binom{p}{0}$ then $\delta_C^{a'}(s) \neq \emptyset$ for all immediate successors a' of a. Note that compatibility depends on our choice for the $next(\cdot)$ function, e.g., P and C would not be compatible if we chose the $next(\cdot)$ function which maps $\bot$ to q.

Extension. Given two compatible non-blocking structures K_1 and K_2, we can prove that immediate successors in $\mathcal{B}$ satisfy the one-step extension criterion, just as in the previous section. Likewise, for every $w \in B$, we can construct a chain $(w_i)_{i \in \mathbb{N}}$ converging to w such that w_{i+1} is an immediate successor of w_i for all $i \in \mathbb{N}$. Therefore, we get the same proposition as for Moore machines.

Proposition 8 *Let K_1, K_2 be compatible non-blocking structures, and let $w \in \mathcal{B}$. Then there exists a chain $(w_i)_{i \in \mathbb{N}}$ with limit w such that $(\!|K_1|\!)$ and $(\!|K_2|\!)$ extend along $(w_i)_{i \in \mathbb{N}}$.*

Assume-Guarantee Rule. Now, Propositions 5, 6, 7, 8 for Kripke structures essentially correspond to Propositions 1, 2, 3, 4 for Moore machines, respectively. By replaying the proof of Theorem 2, we get a rule for the partial trace semantics $(\!|\cdot|\!)$, from which our new assume-guarantee rule for the trace semantics $[\![\cdot]\!]$ of Kripke structures follows.

Theorem 3 *Let P_1, P_2, S_1, S_2 be Kripke structures such that P_1 and P_2 are compatible and non-blocking. Then rule (7) is sound.*

$$\frac{(\!|P_2|\!) \cap (\!|S_1|\!) \subseteq (\!|P_1|\!) \qquad (\!|P_1|\!) \cap (\!|S_2|\!) \subseteq (\!|P_2|\!)}{(\!|S_1|\!) \cap (\!|S_2|\!) \subseteq (\!|P_1|\!) \cap (\!|P_2|\!)} \tag{7}$$

Corollary 2 *Let P_1, P_2, S_1, S_2 be Kripke structures such that P_1 and P_2, P_1 and S_2 as well as S_1 and P_2 all are compatible (w.r.t. the same $next(\cdot)$ function), and P_1 and P_2 are non-blocking. Then rule (8) is sound.*

$$\frac{[\![P_2 \parallel S_1]\!] \subseteq [\![P_1]\!] \qquad [\![P_1 \parallel S_2]\!] \subseteq [\![P_2]\!]}{[\![S_1 \parallel S_2]\!] \subseteq [\![P_1 \parallel P_2]\!]} \tag{8}$$

Proof. We assume the premises of the corollary. By Propositions 6 and 7, the premises of rule (8) are equivalent to $(\!|P_2|\!) \cap (\!|S_1|\!) \subseteq (\!|P_1|\!)$ resp. $(\!|P_1|\!) \cap (\!|S_2|\!) \subseteq (\!|P_2|\!)$. Hence, Theorem 3 implies $(\!|S_1|\!) \cap (\!|S_2|\!) \subseteq (\!|P_1|\!) \cap (\!|P_2|\!)$, which by Proposition 7 (twice) implies $(\!|S_1 \parallel S_2|\!) \subseteq (\!|S_1|\!) \cap (\!|S_2|\!) \subseteq (\!|P_1|\!) \cap (\!|P_2|\!) = (\!|P_1 \parallel P_2|\!)$. By Proposition 6, this is equivalent to the conclusion of rule (8). □

5 Encoding of Mealy Machines into Kripke Structures

Mealy machines are a popular extension of Moore machines where the output function may depend on both, the current state and input. This section briefly sketches how Mealy machines can be encoded into Kripke structures so as to prove that our assume-guarantee rule for Kripke structures generalizes the known rules for Mealy machines.

Mealy Machines. We define Mealy machines with explicit dependencies in the spirit of [3], i.e., a *Mealy machine* M is a seven-tuple $(I, O, S, S^\iota, \delta, \rho, \rhd)$ where I, O, S, S^ι and δ are as for Moore machines, $\rho : S \times \Sigma_\mathrm{t}|_I \to \Sigma_\mathrm{t}|_O$ is the output function, and the dependencies $\rhd \subseteq \mathcal{X} \times \mathcal{X}$ are a strict partial order. M must be non-blocking, and $\rhd$ must be *complete*, meaning that $y \not\rhd x$ implies that the current value of y never depends on the current value of x; formally, if $y \not\rhd x$ for some $y \in O$ and $x \in I$, then for all $s \in S$ and all $a, a' \in \Sigma_\mathrm{t}|_I$, $a|_{I\setminus\{x\}} = a'|_{I\setminus\{x\}}$ implies $\rho(s,a)|_{\{y\}} = \rho(s,a')|_{\{y\}}$.

By $[\![M]\!]$ we denote the set of traces of M, where a word $w = a_1 \dots a_n \in \Sigma_\mathrm{t}^*$ is a *trace* of M if there exists a sequence $(s_1, \dots, s_n)$ of states in S such that $s_1 \in S^\iota$, $s_{i+1} \in \delta(s_i, a_i|_I)$ for all $i < n$, and $a_i|_O = \rho(s_i, a_i|_I)$ for all $i \le n$.

Care must be taken to avoid circular dependencies when defining the product. We call two Mealy machines $M_1 = (I_1, O_1, \dots, \rhd_1)$ and $M_2 = (I_2, O_2, \dots, \rhd_2)$ *compatible* if O_1 and O_2 are disjoint, and $(\rhd_1 \cup \rhd_2)^+$, the transitive closure of $\rhd_1$ and $\rhd_2$, is a strict partial order. In this case, the *product* $M_1 \| M_2 = (I, O, \dots, \rhd)$, where $O = O_1 \cup O_2$, $I = (I_1 \cup I_2) \setminus O$ and $\rhd = (\rhd_1 \cup \rhd_2)^+$, is defined and a Mealy machine. Further details of the product construction are not interesting for our purpose, it suffices to know that $[\![M_1 \parallel M_2]\!] = [\![M_1]\!] \cap [\![M_2]\!]$.

Encoding. We encode the Mealy machine M into $\mathcal{K}(M) = (\bar{S}, \bar{\iota}, \bar{\delta}, \bar{\rho})$, where

$$\begin{aligned}
\bar{S} &= \{\bar{\iota}\} \uplus \{(s,a) \in S \times \Sigma_\mathrm{t} \mid \rho(s, a|_I) = a|_O\},\\
\bar{\delta}(\bar{\iota}) &= (S^\iota \times \Sigma_\mathrm{t}) \cap \bar{S},\\
\bar{\delta}\big((s,a)\big) &= \big(\delta(s, a|_I) \times \Sigma_\mathrm{t}\big) \cap \bar{S}, \text{ and}\\
\bar{\rho}\big((s,a)\big) &= a.
\end{aligned}$$

Obviously, $\mathcal{K}(M)$ is a non-blocking Kripke structure, and $[\![\mathcal{K}(M)]\!] = [\![M]\!]$. Furthermore, given two compatible Mealy machines M_1 and M_2, the Kripke structures $\mathcal{K}(M_1)$ and $\mathcal{K}(M_2)$ are compatible. More precisely, $\mathcal{K}(M_1)$ and $\mathcal{K}(M_2)$ are compatible w.r.t. every $\mathit{next}(\cdot)$ function which meets the requirement that for all $a \in \Sigma \setminus \Sigma_\mathrm{t}$, $\mathit{next}(a)$ is minimal in $\mathcal{X} \setminus \mathit{dom}(a)$ w.r.t. the dependencies $\rhd$ of $M_1 \| M_2$. As a consequence, our encoding distributes over products, i.e., $[\![\mathcal{K}(M_1 \parallel M_2)]\!] = [\![\mathcal{K}(M_1) \parallel \mathcal{K}(M_2)]\!]$ for all compatible Mealy machines M_1 and M_2.

Assume-Guarantee Rule. By specializing our assume-guarantee rule for Kripke structures to Mealy machines, we get essentially the rule [3] presents for Reactive Modules.

Theorem 4 *Let P_1, P_2, S_1, S_2 be Mealy machines such that P_1 and P_2, P_1 and S_2, S_1 and P_2 as well as S_1 and S_2 all are compatible, and the dependencies $\rhd_{P_1,P_2}$ of $P_1 \parallel P_2$ are contained in the dependencies $\rhd_{S_1,S_2}$ of $S_1 \parallel S_2$. Then rule (9) is sound.*

$$\frac{[\![P_2 \parallel S_1]\!] \subseteq [\![P_1]\!] \qquad [\![P_1 \parallel S_2]\!] \subseteq [\![P_2]\!]}{[\![S_1 \parallel S_2]\!] \subseteq [\![P_1 \parallel P_2]\!]} \tag{9}$$

Proof. We assume the premises of the theorem. Via our encoding into Kripke structures, the premises of rule (9) are equivalent to $[\![\mathcal{K}(P_2) \parallel \mathcal{K}(S_1)]\!] \subseteq [\![\mathcal{K}(P_1)]\!]$ resp. $[\![\mathcal{K}(P_1) \parallel \mathcal{K}(S_2)]\!] \subseteq [\![\mathcal{K}(P_2)]\!]$. As $\rhd_{P_1,P_2} \subseteq \rhd_{S_1,S_2}$ implies that the dependencies $\rhd_{P_1,S_2}$ of $P_1 \parallel S_2$ resp. $\rhd_{S_1,P_2}$ of $S_1 \parallel P_2$ are also contained in $\rhd_{S_1,S_2}$, we can choose a $next(\cdot)$ function according to $\rhd_{S_1,S_2}$ as described above, and obtain that $\mathcal{K}(P_1)$ and $\mathcal{K}(P_2)$, $\mathcal{K}(P_1)$ and $\mathcal{K}(S_2)$ as well as $\mathcal{K}(S_1)$ and $\mathcal{K}(P_2)$ all are compatible w.r.t. that $next(\cdot)$ function. Hence, Corollary 2 yields $[\![\mathcal{K}(S_1) \parallel \mathcal{K}(S_2)]\!] \subseteq [\![\mathcal{K}(P_1) \parallel \mathcal{K}(P_2)]\!]$, which is equivalent to the conclusion of rule (9) via our encoding. □

The side condition of Proposition 5 in [3] seems to impose fewer restrictions on the machines than our above theorem; in particular, that proposition does not require the inclusion $\rhd_{P_1,P_2} \subseteq \rhd_{S_1,S_2}$ to hold a priori. However, this inclusion follows from the conclusion of that proposition, so the rule in [3] is not stronger than ours.

6 Conclusion and Future Work

We have presented a circular assume-guarantee rule for a particularly simple modeling formalism, namely downward-closed sets. This simplicity allows a concise inductive proof of the rule, unobscured by technical details of the formalism. Yet, the formalism is so general that our rule can serve as a pattern for assume-guarantee rules in a variety of more complex formalisms. To derive a rule in some particular formalism, it suffices to provide a downward-closed semantics that meets the requirements expressed in the side condition of our generic rule — no need to cope with circular assumptions any more.

Various circular assume-guarantee rules have been proposed for Moore- and Mealy-like formalisms that model systems as transition graphs, for instance L-processes [7], Reactive Modules [3] and Mealy machines [9]. Typically, these formalisms are based on trace containment. By instantiating our framework to Kripke structures, we have derived a new trace-based assume-guarantee rule, which generalizes the rules for trace-based Mealy-like formalisms. It remains to be investigated whether our framework can also be instantiated to formalisms based on tree containment, for instance [5]. Instantiations to process algebras would be interesting, too; in particular, we would like to explain the rather involved rule for CCS in [12].

Assume-guarantee rules are also known for logical formalisms where systems are modeled by logical formulae, e.g., [1] provides a rule for Lamport's TLA and [6] one for LTL. Closest in spirit to our work may be [2] where general logical assume-guarantee rules are derived which embed the circularity-breaking side condition into non-classical logics. In contrast, the goal of our framework is to make this side condition explicit.

The original motivation for this work is the development of an assume-guarantee rule for model checking in a constraint-based setting [4,11]. Our rule for Kripke structures is applicable in this setting, yet checking its side condition, in particular compatibility, may require the examination of infinite state spaces. In a model checking tool, we would like these checks to be performed 'behind the scenes', hence we are looking for constraint-based methods to enhance termination of compatibility checks.

Acknowledgment. The author wishes to thank Witold Charatonik, Andreas Podelski, Sriram K. Rajamani and Jean-Marc Talbot for helpful discussions and comments.

References

1. Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, 1995.
2. Martín Abadi and Gordon D. Plotkin. A logical view of composition. *Theoretical Computer Science*, 114(1):3–30, 1993.
3. Rajeev Alur and Thomas A. Henzinger. Reactive modules. In *Proc. 11th Annual Symposium On Logic In Computer Science*, pages 207–218, 1996.
4. Giorgio Delzanno and Andreas Podelski. Model checking in clp. In *TACAS'99: Tools and Algorithms for the Construction and Analysis of Systems*, Springer LNCS 1579, pages 223–239, 1999.
5. Thomas A. Henzinger, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. An assume-guarantee rule for checking simulation. In *FMCAD'98 : Formal methods in computer-aided design*, Springer LNCS 1522, pages 421–432, 1998.
6. Bengt Jonsson and Yih-Kuen Tsay. Assumption/guarantee specifications in linear-time temporal logic. *Theoretical Computer Science*, 167(1–2):47–72, 1996.
7. Robert P. Kurshan. *Computer-aided verification of coordinating processes*. Princeton University Press, 1994.
8. Patrick Maier. A set-theoretic framework for assume-guarantee reasoning. Technical Report MPI-I-2001-2-002, Max-Planck-Institut für Informatik, 2001.
9. K. L. McMillan. A compositional rule for hardware design refinement. In *CAV'97 : Computer aided verification*, Springer LNCS 1254, pages 207–218, 1997.
10. Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4):417–426, 1981.
11. Andreas Podelski. Model checking as constraint solving. In *SAS 2000 : Static Analysis*, Springer LNCS 1824, pages 22–37, 2000.
12. Sriram K. Rajamani and Jakob Rehof. A behavioral module system for the π-calculus. In *SAS'01 : Static Analysis*, 2001. To appear.

Foundations for Circular Compositional Reasoning

Mahesh Viswanathan[1] and Ramesh Viswanathan[2]

[1] DIMACS & Telcordia Technologies,
Piscataway NJ 08854, USA
maheshv@dimacs.rutgers.edu
[2] Bell Laboratories, Holmdel NJ 07733, USA
rv@research.bell-labs.com

Abstract. Compositional proofs about systems of many components require circular reasoning principles in which properties of other components need to be assumed in proving the properties of each individual component. A number of such circular assume-guarantee rules have been proposed for different concurrency models and different forms of property specifications. In this paper, we provide a framework that unifies and extends these results. We define an assume-guarantee semantics for properties expressible as least or greatest fixed points, and a circular compositional rule that is sound with respect to this semantics. We demonstrate the utility of this general rule by applying it to trace semantics with linear temporal logic specifications, and trace tree semantics with automata refinement specifications. For traces, we derive a new assume-guarantee rule for the "weakly until" operator of linear temporal logic and show that previously proposed assume-guarantee rules can be seen as special instances of our rule. For trace trees, we derive a rule for parallel composition of Moore machines, and show that the rule of [7] is a special instance thus yielding an alternate proof of the results in [7].

1 Introduction

Program verification is concerned with determining whether a formal model of a system satisfies certain correctness properties. The most popular algorithmic technique, model checking, systematically steps through the global states of the system while checking various properties at each stage. However, such a method runs into the well-known problem of *state-space explosion* which severely limits the size of analyzable systems.

The classical solution to this problem is to verify systems compositionally or hierarchically in which suitable properties of individual components of a system are verified in isolation, and these "local" properties are then combined to prove the correctness of the system as a whole. For a system $P = P_1||P_2$, a rule that supports such compositional reasoning might take the form that if P_1 satisfies property φ_1 and P_2 satisfies φ_2, then P satisfies $\varphi_1 \wedge \varphi_2$. While this form of reasoning is sound for various specification languages and concurrency models,

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 835–847, 2001.

it is often not helpful. This is because, in many cases, although the global system P may satisfy the property $\varphi_1 \wedge \varphi_2$, the decomposed proof obligations of P_1 and P_2 satisfying φ_1 and φ_2 may not necessarily hold. Typically, a component P_1 does not satisfy φ_1 "unconditionally" (*i.e.*, when composed with arbitrary components) but only when composed with another component that behaves like P_2.

In the assumption-guarantee paradigm [15] (also referred to as rely-guarantee and assumption-commitment in the literature), each component of a system is specified in terms of assumptions it makes about its environment (or other components), and properties it guarantees about its behavior, provided the assumptions hold. Using $\varphi \triangleright \psi$ to syntactically denote the property that under the assumptions φ, the property ψ is guaranteed, a *circular* compositional rule that one would like for the system $P = P_1||P_2$ would be that if P_1 satisfies the property $\varphi_2 \triangleright \varphi_1$ and P_2 satisfies the property $\varphi_1 \triangleright \varphi_2$ then P satisfies $\varphi_1 \wedge \varphi_2$. However, because of the circularity of each component making assumptions about the other component's yet-to-be proven guarantees, such rules are hard to construct and are in fact sound only for some special classes of properties. For example, such a rule is clearly not propositionally valid if we interpret assumption-guarantee to be propositional implication. That such a circular argument is indeed sound under some conditions, was first observed by Misra and Chandy [13], and later formalized by Abadi and Lamport [1,2]. Subsequently, circular compositional rules have been proposed for a variety of computational models. Traces and safety properties are considered in [13,8,1,2,14,10,6,9,3,4,11,5], and more recently, McMillan [12] has obtained a circular reasoning principle for certain liveness properties. This approach for traces has been extended to trace trees in which specifications are given via some form of automata-refinement [3,4,7]. Henzinger, Qadeer, Rajamani, and Tasiran [7] present such an assumption/guarantee rule for Moore machines with synchronous parallel composition. Liveness requirements can be specified in terms of additional Streett fairness conditions imposed on Moore and Mealy machines.

In this paper, we present a common unifying framework and reasoning rule which encompasses previously proposed circular composition rules and which can be applied to derive circular composition rules for more expressive classes of properties than have previously been considered. Our starting point is the well known observation that many important properties, including safety and liveness properties, can be formulated as least or greatest fixed points of appropriate functions. We therefore define a semantics for assume-guarantee specifications in which the assumptions and guarantees are least or greatest fixed points. We then formulate a rule for composing such local assume-guarantee specifications to derive a global assume-guarantee specification that is sound with respect to this semantics. Our assume-guarantee semantics and rule are formulated and proved sound in a general setting which requires limited assumptions about the semantics of processes or the form of specifications We next show how this general rule can be fruitfully applied to some specific contexts. First, we consider the setting where semantics for processes are defined as traces and specifications as

linear temporal logic formulas. Here, we derive, from our general framework, an assume-guarantee rule for the "weakly until" temporal logic operator, and show that it generalizes the most powerful rule previously known in this setting [12]. The second setting we consider is that of Moore machines interpreted in a semantic space consisting of labelled trace trees. We derive an assume-guarantee rule for synchronous parallel composition of Moore machines, and show that this rule subsumes the strongest known circular rule for Moore machines, that of [7]. Together, the two settings demonstrate that our general theorem unifies and can be applied to disparate process semantics and disparate specification formulations.

The rest of the paper is organized as follows. In Section 2, we present our semantics for assumption/guarantee specifications for fixed point properties, and general rules for composing such specifications. Section 3 applies these rules to traces and linear temporal logic, and Section 4 develops the results for Moore machines. We make concluding remarks in Section 5.

2 Assume-Guarantee Rule for Fixed Points

2.1 Preliminaries and Notation

Let $\mathcal{C}$ be a set; intuitively, $\mathcal{C}$ represents the semantic space of all valid computations (such as traces or trees). Thus, the semantics of a program or automaton P, written $[\![P]\!]$, will be a subset of $\mathcal{C}$. However, the framework and rule we present in this section will require no assumptions on the structure of computations and we therefore leave the elements of $\mathcal{C}$ uninterpreted. We use $\sigma, \sigma_1, \ldots$ to range over computations, *i.e.*, $\sigma, \sigma_1, \ldots \in \mathcal{C}$ and a specification $S \subseteq \mathcal{C}$. A computation σ satisfies a specification S, written $\sigma \models S$, if and only if $\sigma \in S$, and a program P satisfies a specification S, $P \models S$, if and only if $\forall \sigma \in [\![P]\!].\ \sigma \models S$.

We are interested in properties that are recursive or fixed point specifications. Writing $\mathcal{P}(\mathcal{C})$ to denote the power set of $\mathcal{C}$, let $F : \mathcal{P}(\mathcal{C}) \to \mathcal{P}(\mathcal{C})$ be a monotonic operator with respect to the subset ordering, $\subseteq$, on $\mathcal{P}(\mathcal{C})$. By the Tarski-Knaster theorem [17], F has both greatest and least fixed points. We will use $\nu(F)$ to denote the greatest fixed point of F, $\mu(F)$ for its least fixed point, and meta-variables $\rho, \rho_1, \ldots$ to stand for either μ or ν so that $\rho(F)$ will denote one of the two fixed points of F. The Tarski-Knaster construction approximates the fixed points through repeated iterations of F whose limit yields the desired fixed point. Let $\omega = \{0, 1, 2, \ldots\}$ be the set of all natural numbers; the k'th approximation of the fixed point $\rho(F)$, which we denote $[\rho(F)]^k$, is defined by induction:

$$\begin{array}{llll} [\mu(F)]^0 & = \emptyset & [\nu(F)]^0 & = \mathcal{C} \\ [\mu(F)]^{k+1} & = F([\mu(F)]^k) & [\nu(F)]^{k+1} & = F([\nu(F)]^k) \\ [\mu(F)]^{\omega} & = \bigcup_{k \in \omega} [\mu(F)]^k & [\nu(F)]^{\omega} & = \bigcap_{k \in \omega} [\nu(F)]^k \end{array}$$

Depending on the cardinality of S, this process of constructing progressively better approximations needs to be continued for transfinite ordinals before it stabilizes to the fixed point. However, all applications considered in this paper

converge to their fixed points within ordinal ω; we say that a fixed point specification $\rho(F)$ (where ρ is μ or ν) is ω-*convergent* if $F([\rho(F)]^\omega) = [\rho(F)]^\omega$. For example, if F is ω-continuous, *i.e.*, for any ω-increasing chain $\{S_i \mid i \in \omega\}$ with $S_i \subseteq S_{i+1}$ for all i, we have that $F(\cup_i S_i) = \cup_i F(S_i)$, then $\mu(F)$ is ω-convergent. Similarly, if F is ω-cocontinuous, then $\nu(F)$ is ω-convergent. Thus, ω-continuity and ω-cocontinuity provide sufficient checks for ω-convergence of $\mu(F)$ and $\nu(F)$ respectively.

2.2 Semantics of Assume-Guarantee Specifications

We are now ready to define the semantics for assume-guarantee specifications, which we syntactically denote by $\rho(A) \triangleright \rho'(G)$, whose informal reading is that the guarantee specification $\rho'(G)$ is satisfied whenever the assumption specification $\rho(A)$ is satisfied. Classically, in the context of safety properties, it has been interpreted to require the guarantee to be satisfied upto time instant $k+1$ whenever the assumption is satisfied upto time instant k. Our definition is motivated by this classical notion with the key insight being that the k'th approximation provides the suitable analogue of "upto time instant k" . Further, to allow the weakest condition necessary for the guarantee to be satisfied when it is a least fixed point, we generalize the condition on the guarantee specification requiring it to be satisfied at any future approximation (rather than the immediately next one).

Definition 1 (Assume-Guarantee Semantics).

$$\sigma \models \rho(A) \triangleright \rho'(G) \quad \textit{iff} \quad \forall k \geq 0.\ \sigma \models [\rho(A)]^k \ \Rightarrow\ \exists k' > k.\ \sigma \models [\rho'(G)]^{k'}$$

An immediate consequence of our assume-guarantee semantics is that it generalizes propositional implication.

Proposition 1. *Suppose that the fixed point specification $\rho'(G)$ is ω-convergent. Then the following rule is sound*

$$\frac{\sigma \models \rho(A) \triangleright \rho'(G)}{\sigma \models \rho(A) \ \Rightarrow\ \sigma \models \rho'(G)} \qquad (Imp\ \triangleright)$$

When the consequent of an assume-guarantee specification is a greatest fixed point, then our general definition reduces to the more familiar classical notion. When the consequent is a least fixed point, then our semantics places no stronger requirements than that the least fixed point be satisfied.

Proposition 2. *1.* $\sigma \models \rho(A) \triangleright \nu(G)$ *iff* $\forall k \geq 0.\ \sigma \models [\rho(A)]^k \Rightarrow \sigma \models [\nu(G)]^{k+1}$
2. $\sigma \models \rho(A) \triangleright \mu(G) \quad \textit{iff} \quad \forall k \geq 0.\ (\sigma \models [\rho(A)]^k \ \Rightarrow\ \exists k' \geq 0.\ \sigma \models [\mu(G)]^{k'})$

2.3 Composing Assume-Guarantee Specifications

In this section, we describe inference rules for composing assume-guarantee specifications. We first present general rules that are applicable to arbitrary assume-guarantee specifications without making any assumptions about the form of fixed points. We next specialize these general rules to derive more powerful rules that are applicable when the assumption is a greatest fixed point.

General Fixed Points. The general semantic rule prescribes conditions under which it is sound to compose two local behaviors, $\sigma_1 \models \rho(A_1) \triangleright \rho'(G_1)$ and $\sigma_2 \models \rho(A_1) \triangleright \rho'(G_2)$ to derive the behavior of the global system $\sigma \models \rho(A) \triangleright \rho'(G)$ (in typical applications of the rule, σ will be the parallel composition of σ_1 and σ_2). The conditions can be intuitively read as follows: *(1a), (1b)* If the global assumption is satisfied currently and one of the local guarantees is satisfied eventually, then the other local assumption is satisfied eventually. *(2)* If the local guarantees hold then the global guarantee holds eventually. *(3)* If the global assumption is satisfied then one of the local guarantees holds eventually. Conditions *(1a), (1b)* allow each local guarantee to be used circularly in satisfying the other's local assumptions, and Condition *(3)* breaks this circularity to ensure soundness.

Theorem 1 (Semantic Rule for General Fixed Points). *The following rule, (Comp $\rho \triangleright \rho'$), is sound:*

$$\frac{\begin{array}{l}\sigma_1 \models \rho(A_1) \triangleright \rho'(G_1) \qquad \sigma_2 \models \rho(A_2) \triangleright \rho'(G_2) \\ (1a)\ \forall k, k' \geq k.\ \sigma \models [\rho(A)]^k \wedge \sigma_2 \models [\rho'(G_2)]^{k'} \Rightarrow \exists k'' \geq k.\ \sigma_1 \models [\rho(A_1)]^{k''} \\ (1b)\ \forall k, k' \geq k.\ \sigma \models [\rho(A)]^k \wedge \sigma_1 \models [\rho'(G_1)]^{k'} \Rightarrow \exists k'' \geq k.\ \sigma_2 \models [\rho(A_2)]^{k''} \\ (2)\ \ \forall k.\ \sigma_1 \models [\rho'(G_1)]^k \wedge \sigma_2 \models [\rho'(G_2)]^k \Rightarrow \exists k' \geq k.\ \sigma \models [\rho'(G)]^{k'} \\ (3)\ \ \forall k.\ \sigma \models [\rho(A)]^k \Rightarrow \exists k' \geq k.\ \sigma_1 \models [\rho'(G_1)]^{k'} \vee \sigma_2 \models [\rho'(G_2)]^{k'}\end{array}}{\sigma \models \rho(A) \triangleright \rho'(G)}$$

Observe that, the Rule (*Comp* $\rho \triangleright \rho'$) requires that the assumptions in the assume-guarantee specifications of σ_1, σ_2, and σ be the same form of fixed point (ρ) and similarly for the guarantees. However, certain variations are sound — we formally allow them using a subsumption rule, for which we first define a subtyping relation.

Definition 2. *For monotonic operators $F, G : \mathcal{P}(\mathcal{C}) \to \mathcal{P}(\mathcal{C})$, we say that $F \ll G$ iff for all $S \subseteq \mathcal{C}$ we have that $F(S) \subseteq G(S)$. We say that a fixed point specification $\rho'(F')$ is a subtype of another fixed point specification $\rho(F)$, written $\rho'(F') <: \rho(F)$, if and only if the following conditions hold:*

(1) Either $\rho' = \mu$ and $\rho = \nu$, or $\rho' = \rho$
(2) $F' \ll F$.

Theorem 2 (Subsumption). *The following rule is sound:*

$$\frac{\sigma \models \rho_1(A) \triangleright \rho_2(G) \qquad {\rho'}_1(A') <: \rho_1(A) \qquad \rho_2(G) <: {\rho'}_2(G')}{\sigma \models \rho'_1(A') \triangleright \rho'_2(G')} \qquad (Sub\ \triangleright)$$

Now, given two local assumption-guarantee specifications in which the forms of the fixed points in the assumptions (or guarantees) do not match, we can use Rule $(Sub\ \triangleright)$ on one or both of them to transform to a form in which the corresponding fixed points match and Rule $(Comp\ \rho \triangleright \rho')$ is applicable. Having derived the global behavior, a further use of subsumption allows one to change the form of the fixed point in the assumption or guarantee (but only in a manner consistent with subtyping).

Greatest Fixed Point Assumptions. The true payoff of the general inference rules established in Section 2.3 occurs when the assumption specifications are greatest fixed points. In particular, we can support truly circular reasoning without having to explicitly break the circularity (through establishing Condition *(3)*). To this end, we first prove the following lemma.

Lemma 1. *Suppose that* $\sigma_i \models \nu(A_i) \triangleright \rho(G_i)$ *for* $i = 1, 2$ *and that*

$$\forall k, k' \geq k. \sigma \models [\nu(A)]^k \wedge \sigma_1 \models [\rho(G_1)]^{k'} \wedge \sigma_2 \models [\rho(G_2)]^{k'} \Rightarrow \sigma_1 \models [\nu(A_1)]^k \wedge \sigma_2 \models [\nu(A_2)]^k$$

Then we have that for any k,

$$\sigma \models [\nu(A)]^k \Rightarrow \exists k' \geq k.\ \sigma_1 \models [\rho(G_1)]^{k'} \wedge \sigma_2 \models [\rho(G_2)]^{k'}$$

Lemma 1 is established by induction on k, and the induction step makes crucial use of the fact that in our semantics of assumption guarantee-semantics we require the guarantee to hold at a *strictly greater* approximation. It therefore illustrates the precise origin of the condition $k' > k$ (rather than $k' \geq k$) in Definition 1.

Using Lemma 1, we can instantiate Rule $(Comp\ \rho \triangleright \rho')$ to derive a stronger rule, $(Comp\ \nu \triangleright \nu)$, for greatest fixed points. First, the stronger rule does not require Condition *(3)* to be established. Second, in establishing the local assumptions (Conditions *(1a)* and *(1b)*), we are now allowed to assume *both* local guarantees and that they hold *currently* (rather than eventually). Rule $(Conj\ \nu)$ is a further application of $(Comp\ \nu \triangleright \nu)$, which illustrates the circular reasoning supported by Rule $(Comp\ \nu \triangleright \nu)$ more clearly.

Corollary 1 (Semantic Rule for Greatest Fixed Points). *The following rules,* $(Comp\ \nu \triangleright \rho)$ *and* $(Conj\ \nu)$, *are sound:*

$$\frac{\begin{array}{c}\sigma_1 \models \nu(A_1) \triangleright \nu(G_1) \qquad \sigma_2 \models \nu(A_2) \triangleright \nu(G_2) \\ (1)\ \forall k.\ \sigma \models [\nu(A)]^k \wedge \sigma_1 \models [\nu(G_1)]^k \wedge \sigma_2 \models [\nu(G_2)]^k \Rightarrow \\ \sigma_1 \models [\nu(A_1)]^k \wedge \sigma_2 \models [\nu(A_2)]^k \\ (2)\ \forall k. \sigma_1 \models [\nu(G_1)]^k \wedge \sigma_2 \models [\nu(G_2)]^k \Rightarrow \sigma \models [\nu(G)]^k\end{array}}{\sigma \models \nu(A) \triangleright \nu(G)}$$

$$\frac{\sigma \models \nu(F_1) \triangleright \rho(F_2) \qquad \sigma \models \nu(F_2) \triangleright \rho(F_1)}{\sigma \models \rho(F_1) \wedge \sigma \models \rho(F_2)} \qquad (Conj\ \ \nu)$$

Rule $(Conj\ \ \nu)$ illustrates the importance of the distinction between $\triangleright$ and propositional implication. Had we interpreted $\triangleright$ to be propositional implication, the rule would not have been sound.

In many applications, specifications are most easily described through mutual recursion and we conclude this section by briefly addressing this. Suppose that $F_0, \ldots, F_{n-1} : \mathcal{P}(\mathcal{C})^n \to \mathcal{P}(\mathcal{C})$ are monotonic operators. We write $\rho_i(F_0, \ldots, F_{n-1})$ to denote the i'th solution, where as usual, ρ can be μ or ν. The defining property of these solutions is that

$$\rho_i(F_0, \ldots, F_{n-1}) = F_i(\rho_0(F_0, \ldots, F_{n-1}), \ldots, \rho_{n-1}(F_0, \ldots, F_{n-1}))$$

We define the k'th approximation of such mutually dependent fixed point specifications as follows:

$$\begin{aligned} [\rho_i(F_0, \ldots, F_{n-1})]^0 &= \begin{cases} \emptyset \text{ if } \rho = \mu \\ \mathcal{C} \text{ if } \rho = \nu \end{cases} \\ [\rho_i(F_0, \ldots, F_{n-1})]^{k+1} &= F_i([\rho_0(F_0, \ldots, F_{n-1})]^k, \ldots, [\rho_{n-1}(F_0, \ldots, F_{n-1})]^k) \end{aligned}$$

Having defined the k'th approximations, assume-guarantee specifications involving mutually dependent fixed points are interpreted as in Definition 1. The semantic rule $(Comp\ \ \rho \triangleright \rho')$ is then sound. The semantic rule $(Comp\ \ \nu \triangleright \nu)$ is sound when both assumptions and guarantees are of the form ν_i.

3 Assumption-Guarantee for Traces Semantics

In this section, we consider the framework of linear time temporal logic and traces. Using our general assume-guarantee semantics, we obtain the definition of assume-guarantee specifications for traces, and then derive composition rules for linear time temporal logic. In particular, we derive a circular reasoning principle for "weak until" specifications which are a generalization of properties defined using the "always" modality. Finally, we show that the reasoning principle proposed in [12] is a special case of our rule for until properties.

We take the semantic space of computations $\mathcal{C}$ to be the set of functions $\sigma : \omega \to \mathcal{P}(\text{Prop})$, where Prop is a set of propositions. In other words, computations are viewed as infinite sequences of subsets of propositions. For a computation σ, we use σ_i to denote its i'th element $\sigma(i)$. The suffix of a computation starting at the i'th position will be denoted as $\sigma|_i$; so $\sigma|_i(k) = \sigma(i+k)$. Following [16,1,2,9,11,12], we do not define a syntactic computational model such as a process or automaton; instead a program will simply be a collection of computations. It is often convenient to express such a process as a linear temporal logic property.

A formula in temporal logic is described by the following BNF grammar.

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \circ \varphi \mid \varphi \ \mathrm{U} \ \varphi \mid \varphi \ \mathrm{U_W} \ \varphi$$

where $p \in \text{Prop}$ is a proposition. Apart from classical conjunction and disjunction, the logic includes the modal connectives "next" ($\circ$), "until" (U), and "weak-until" (U_W). Formally, we define a set of computations $[\![\varphi]\!]$ such that $\sigma \models \varphi$ if and only if $\sigma \in [\![\varphi]\!]$.

$$[\![p]\!] = \{\sigma \mid p \in \sigma_0\} \qquad [\![\neg p]\!] = \{\sigma \mid p \notin \sigma_0\}$$
$$[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!] \qquad [\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$$
$$[\![\circ \varphi]\!] = \{\sigma \mid \sigma|_1 \in [\![\varphi]\!]\}$$
$$[\![\varphi \ \text{U} \ \psi]\!] = \mu(F[[\![\varphi]\!], [\![\psi]\!]]) \qquad [\![\varphi \ \text{U}_\text{W} \ \psi]\!] = \nu(F[[\![\varphi]\!], [\![\psi]\!]])$$

where $F[S_1, S_2]$ is the following monotonic operator:

$$F[S_1, S_2](S) = \{\sigma \mid \sigma \in S_2 \text{ or } (\sigma \in S_1 \text{ and } \sigma|_1 \in S)\}$$

As usual, $\Diamond \varphi$ (eventually φ) is equivalent to (true U φ), and $\Box \varphi$ (always φ) is equivalent to (φ U_W false).

Example 1. **[Assume-Guarantee Semantics for Safety]** Consider the property $\Box p$, where p is a proposition. As seen above, $\Box p = \nu(F)$ for the operator,

$$F(S) = \{\sigma \mid p \in \sigma_0 \text{ and } \sigma|_1 \in S\}$$

Now $\sigma \in [\nu(F)]^k$ if and only if $p \in \sigma_i$ for every $0 \leq i \leq k-1$. Hence for an assume-guarantee specification $S = \Box p \triangleright \Box q$ (where q is also a proposition), a computation σ satisfies S if and only if $q \in \sigma_0$ and if $p \in \sigma_i$, for $i \leq k-1$, then $q \in \sigma_j$, for $j \leq k$.

The above discussion can be generalized to any safety property. Consider formulas $\Box \varphi$ and $\Box \psi$, where φ and ψ are past time properties. A computation $\sigma \models \Box \varphi \triangleright \Box \psi$ if and only if σ_0 satisfies ψ and if any prefix $\sigma_0.\sigma_1 \ldots \sigma_{k-1}$ of σ satisfies φ then $\sigma_0.\sigma_1 \ldots \sigma_{k-1}.\sigma_k$ satisfies ψ. Thus, for safety properties our definition for assume-guarantee coincides with the classical definition in [1].

We apply Rule (*Comp* $\nu \triangleright \nu$) to derive a rule for specifications whose outermost connective is U_W. Theorem 3, stated below, highlights the pay-off when the general rule (*Comp* $\nu \triangleright \nu$) is instantiated for a specific computational framework: premises about computations and arbitary fixed point approximations are instead replaced by logical tautologies in Rule (*Comp* $\text{U}_\text{W} \triangleright \text{U}_\text{W}$).

Theorem 3. *The following rule,* (*Comp* $\text{U}_\text{W} \triangleright \text{U}_\text{W}$)*, is sound.* [1]

$$\frac{\begin{array}{ll} \sigma \models (\varphi_1^1 \ \text{U}_\text{W} \ \varphi_2^1) \triangleright (\psi_1^1 \ \text{U}_\text{W} \ \psi_2^1) & \sigma \models (\varphi_1^2 \ \text{U}_\text{W} \ \varphi_2^2) \triangleright (\psi_1^2 \ \text{U}_\text{W} \ \psi_2^2) \\ (1a)\ (\varphi_1 \wedge \psi_1^1 \wedge \psi_1^2) \models (\varphi_1^1 \wedge \varphi_1^2) & (1b)\ (\varphi_2 \wedge \psi_2^1 \wedge \psi_2^2) \models (\varphi_2^1 \wedge \varphi_2^2) \\ (2a)\ (\psi_1^1 \wedge \psi_1^2) \models \psi_1 & (2b)\ (\psi_2^1 \wedge \psi_2^2) \models \psi_2 \end{array}}{\sigma \models (\varphi_1 \ \text{U}_\text{W} \ \varphi_2) \triangleright (\psi_1 \ \text{U}_\text{W} \ \psi_2)}$$

[1] Recall, $\varphi \models \psi$ iff for every σ, $\sigma \models \varphi$ implies $\sigma \models \psi$.

McMillan's rule for liveness. McMillan [12] presents a rule for reasoning about $\Box\,\varphi$ properties. His rule says that if a computation satisfies a collection of specially constructed until properties built up from formulas in some set P, then the computation satisfies $\Box\,\varphi$ for every $\varphi \in P$. We make the key observation that any computation satisfying the until properties in McMillan's theorem also satisfies an assumption-guarantee specification of $\Box\,\varphi$ formulas. More precisely, we show that if $\sigma \models \neg(\varphi\ \mathrm{U}\ \psi)$ then $\sigma \models \Box\,\varphi \triangleright \Box\,(\neg\psi)$. Since properties of the form $\Box\,\varphi$ are a special class of weak until specifications, the soundness of McMillan's rule then follows by repeated applications of Rule $(Comp\ \mathrm{U_W} \triangleright \mathrm{U_W})$.

Theorem 4. *Given a set P of formulas, a well founded order $\prec$ on P, and for all $\varphi \in P$, sets $\Theta_\varphi \subseteq \Delta_\varphi \subseteq P$, such that $\psi \in \Theta_\varphi$ implies $\psi \prec \varphi$. If $\sigma \models \neg(\Delta_\varphi\ \mathrm{U}\ (\Theta_\varphi \wedge \neg\varphi))$, for all $\varphi \in P$, then $\sigma \models \Box\,\varphi$, for all $\varphi \in P$.*

4 Assume-Guarantee for Tree Semantics

In this section, we consider a semantic setting of trace trees with Moore machines defining our programming model. We derive a composition rule, $(Comp\ \ Moore)$, for Moore machines with synchronous parallel composition and show how the rule for simulation preorder in [7] is a special instance of $(Comp\ \ Moore)$. For details on the advantages of trace-trees and using automata-refinement specifications, we refer the reader to [7]. Finally, we note that while our results are developed for Moore machine, they apply as well to other non-blocking, finitely non-deterministic, receptive models such as Reactive Modules [4], where parallel composition corresponds to logical conjunction.

4.1 Moore Machines and Their Semantics

Preliminaries. A (rooted) tree is a set $\tau \subseteq \omega^*$ such that if $xn \in \tau$, for $x \in \omega^*$ and $n \in \omega$, then $x \in \tau$ and $xm \in \tau$ for all $0 \leq m \leq n$. The elements of τ represent nodes: the empty string ϵ is the root, and for each node x, the nodes of the form xn, for $n \in \omega$, are the children of x. The edges of the tree are pairs $\langle x, xn\rangle$, where $x, xn \in \tau$. The number of children of a node x is degree of the node, and it is denoted by $deg(x)$. A tree τ is finitely branching if for every $x \in \tau$, $deg(x)$ is finite. A tree τ is finite if the set τ is finite; otherwise τ is said to be infinite. The set of finite branching, infinite trees is denoted by $\mathbf{T}^\omega_f$. A node $x \in \tau$ is said to be at depth $|x|$, where $|x|$ denotes the length of string x. For a tree τ, the subtree rooted at $x \in \tau$, is the tree $\tau|_x = \{i \mid xi \in \tau\}$.

Given sets A and B, an $\langle A, B\rangle$-labeled tree is a triple $\hat{\tau} = \langle \tau, \lambda, \delta\rangle$, where τ is a tree, $\lambda : \tau \to A$ is a labeling function that that maps each node of the tree to an element of A, and $\delta : \tau \times \omega \to B$ is a function that labels each edge $\langle x, xn\rangle$ in τ with $\delta(x, n) \in B$. Given a labeled tree $\hat{\tau}$, the labeled tree rooted at x is $\hat{\tau}|_x = \langle \tau|_x, \lambda', \delta'\rangle$, where $\lambda'(y) = \lambda(xy)$, and $\delta'(y, n) = \delta(xy, n)$.

Definition 3. *A Moore machine is a tuple $P = \langle S, s_0, I, O, L, R\rangle$ where, S is the set of states, $s_0 \in S$ is the initial state, I is the set of input propositions, O*

is the set of output propositions disjoint from I, $L : S \to \mathcal{P}(O)$ *is a function that labels each state with the set of output propositions true in that state, and* $R \subseteq S \times \mathcal{P}(I) \times S$ *is the transition relation. We write* $R(s, i, t)$ *instead of* $(s, i, t) \in R$.

In what follows we only consider Moore machines that are *non-blocking* and *finitely nondeterministic*. We denote the *parallel composition* of machines P and Q by $P||Q$, and write $P \preceq_s Q$ to denote that a machine Q *simulates* machine P. The definitions of non-blocking, finitely nondeterminitic, parallel composition, and simulation are as in [7].

Semantic Space. We will define the semantics of a Moore machine in terms of labeled, finitely branching infinite trees. A run tree of a machine P is a $\langle S, \mathcal{P}(I)\rangle$-labeled tree $\langle \tau, \lambda, \delta\rangle$, such that $\tau \in \mathbf{T}^{\omega}_{f}$, $\lambda(\epsilon) = s_0$, and for every edge $\langle x, xn\rangle$ of τ we have $R(\lambda(x), \delta(x, n), \lambda(xn))$. Note that since our machines are non-blocking, every machine has at least one run tree. A trace tree $\langle \hat{\tau}, \langle O, I\rangle\rangle \in \mathcal{C}$ of P is a $\langle \mathcal{P}(O), \mathcal{P}(I)\rangle$-labeled tree $\hat{\tau} = \langle \tau, \lambda, \delta\rangle$ such that there is a run tree $T_r = \langle \tau, \lambda', \delta\rangle$ such that for all $x \in \tau$, $\lambda(x) = L(\lambda'(x))$. The semantics of a machine P, denoted by $[\![P]\!]$, is the collection of all trace trees of P.

An alphabet $\langle O, I\rangle$ is said to refine another alphabet $\langle O', I'\rangle$ (denoted as $\langle O, I\rangle \preceq \langle O', I'\rangle$) if $O' \subseteq O$ and $I' \subseteq (I \cup O)$. Consider $\langle O, I\rangle \preceq \langle O', I'\rangle$ and a computation $\sigma = \langle\langle \tau, \lambda, \delta\rangle, \langle O, I\rangle\rangle$. The projection of σ onto $\langle O', I'\rangle$ is $[\sigma]_{\langle O', I'\rangle} = \langle\langle \tau, \lambda', \delta'\rangle, \langle O', I'\rangle\rangle$, where for all $x \in \tau$, $\lambda'(x) = \lambda(x) \cap O'$, and for every edge $\langle x, xn\rangle$ of τ, $\delta'(x, n) = (\delta(x, n) \cup \lambda(x)) \cap I'$.

Under suitable projections, parallel composition of Moore machines corresponds to the "intersection" of the set of trace trees for the component machines, while simulation corresponds to trace tree containment. This is formally stated and proved in Propositions 1 and 2 in [7].

4.2 Assume-Guarantee for Moore Machines

We will now present a compositional reasoning principle for Moore machines that allows one to compose specifications that contain both safety and liveness constraints. As we shall later show, this rule is more general than previous composition rules for Moore machines.

We will call a set $T \subseteq \mathcal{C}$ projection closed if for any $\sigma \in \mathcal{C}$, with alphabet $\langle O, I\rangle$, $\sigma \in T$ if and only if for σ' such that $[\sigma']_{\langle O,I\rangle} = \sigma$, $\sigma' \in T$. In other words, a set is projection closed if it contains the projections of all of its elements and in addition, contains the computations whose projections are in the set. An operator $F : \mathcal{C} \to \mathcal{C}$ is projection preserving if for any projection closed set T, $F(T)$ is also projection closed.

Theorem 5. *For Moore machines* P_1 *and* P_2 *such that* $P_1||P_2$ *exists, and projection preserving operators* F *and* G*, the following rule is sound.*

$$\frac{P_1 \models \nu(F) \triangleright \rho(G) \qquad P_2 \models \nu(G) \triangleright \rho(F)}{P_1||P_2 \models \rho(F) \wedge \rho(G)} \qquad (Comp\ \ Moore)$$

Theorem 5 illustrates how our general compositional rules can be applied to a specific computational model to obtain circular reasoning principles. Conditions (*1*) and (*2*) in the general rule (*Comp* $\nu \triangleright \nu$) can be seen as describing the relationship between the composition of computations and assume-guarantee specifications. They are, therefore, often eliminated when a specific model is considered.

4.3 Compositional Reasoning for Simulation

We now show how the simulation pre-order rule of [7] can be derived from (*Comp Moore*). We begin by defining a natural generalization of projection, which we call observational equivalence. The formal relationship between observational equivalence and projection is captured in Proposition 3.

Definition 4. *Trace trees* $\sigma = \langle\langle \tau, \lambda, \delta\rangle, \langle O, I\rangle\rangle$ *and* $\sigma' = \langle\langle \tau, \lambda', \delta'\rangle, \langle O', I'\rangle\rangle$ *are said to be observationally equivalent, written* $\sigma \approx \sigma'$*, if and only if for every* $x \in \tau$ *and edge* $\langle x, xn\rangle$ *in* τ*, the following two conditions hold:*

(1) $\lambda(x) \cap O' = \lambda'(x) \cap O$*, and*
(2) $(\delta(x,n) \cup \lambda(x)) \cap (O' \cup I') = (\delta'(x,n) \cup \lambda'(x)) \cap (O \cup I)$.

If conditions (1) and (2) hold only for nodes $x \in \tau$ *such that* $|x| < k$ *then we say* σ *and* σ' *are observationally equivalent upto depth* k*, denoted by* $\sigma \approx_{<k} \sigma'$.

Proposition 3. *Let* $\sigma = \langle\langle \tau, \lambda, \delta\rangle, \langle O, I\rangle\rangle$*,* $\sigma' = \langle\langle \tau, \lambda', \delta'\rangle, \langle O', I'\rangle\rangle$*, and* $\sigma'' = \langle\langle \tau, \lambda'', \delta''\rangle, \langle O'', I''\rangle\rangle$ *be trace trees.*

1. *If* $\langle O, I\rangle \preceq \langle O', I'\rangle$ *then* $\sigma \approx \sigma'$ *iff* $[\sigma]_{\langle O', I'\rangle} = \sigma'$.
2. $\sigma' \approx \sigma''$ *iff there is some* σ *such that* $\sigma' = [\sigma]_{\langle O', I'\rangle}$ *and* $\sigma'' = [\sigma]_{\langle O'', I''\rangle}$.

To apply our general assumption-guarantee rule, we recast simulation as a fixed-point definition.

Definition 5. *Consider machine* $P = \langle S, s_0, I, O, L, R\rangle$*, with* $S = \{s_0, s_1, \ldots s_n\}$*. For sets* $Z_0, \ldots, Z_n \subseteq \mathcal{C}$*, define* $F_i(Z_0, \ldots, Z_n)$ *to be the set of all computations* $\sigma = \langle \hat{\tau}, \langle O', I'\rangle\rangle$ *such that for all* $i \in \hat{\tau}$*, we have a* $j \leq n$ *with the following conditions holding: (1)* $\sigma|_i \in Z_j$*, (2)* $\lambda(\epsilon) \cap O = L(s_i) \cap O'$*, and (3)* $(\delta(\epsilon, i) \cup \lambda(\epsilon)) \cap (O \cup I) = (k \cup L(s_i)) \cap (O' \cup I')$*, where* $R(s_i, k, s_j)$*. The specification* $\Phi(P)$ *is defined to be* $\nu_0(F_0, \ldots, F_n)$.

In Definition 5, each F_i constructs trace trees that are observationally equivalent to computations that start in state s_i of machine P. Since simulation corresponds to trace tree containment, the mutually recursive fixed point $\Phi(P)$ captures the class of all machines Q such that $Q \preceq_s P$. These intuitions are captured in the following proposition and corollary.

Proposition 4. $\sigma \models [\Phi(P)]^k$ *iff for some* $\sigma' \in [\![P]\!]$*,* $\sigma \approx_{<k} \sigma'$.

Corollary 2. *1.* $\sigma \models \Phi(P)$ *iff for some* $\sigma' \in [\![P]\!]$, $\sigma \approx \sigma'$.
2. If $P' \preceq_s P$ *then* $P' \models \Phi(P)$.

Having defined the fixed point specification $\Phi(P)$, the final step is to characterize the relationship between parallel composition and our assumption-guarantee semantics. Indeed, subject to some technical conditions on the alphabets of the machines, $P||Q \preceq_s P'$ can be recast as $P \models \Phi(Q) \triangleright \Phi(P')$.

Lemma 2. *Consider machines* P, Q, *and* P' *with alphabets* $\langle O_P, I_P\rangle, \langle O_Q, I_Q\rangle$, *and* $\langle O_{P'}, I_{P'}\rangle$ *respectively such that* $P||Q$ *exists. Furthermore, let* $O_{P'} \subseteq O_P$ *and* $I_{P'} \subseteq (O_P \cup I_P) \cup (O_Q \cup I_Q)$. *Then* $P||Q \preceq_s P'$ *iff* $P \models \Phi(Q) \triangleright \Phi(P')$.

Lemma 2 follows from Proposition 4 and Corollary 2 which establish the relationship between approximations of $\Phi(P)$ and the simulation relation $\preceq_s$; the proof also relies on the machines being non-blocking to "extend" any correct finite trace tree. Theorem 6 of [7] now immediately follows from Lemma 2 by application of rule (*Comp Moore*).

Theorem 6 ([7]). *Consider Moore machines* P, Q, P', Q' *such that* $P||Q$ *and* $P'||Q'$ *exist. Suppose that* $P||Q' \preceq_s P'$ *and* $P'||Q \preceq_s Q'$. *Furthermore, assume that every input of* $P'||Q'$ *is either an input or output of* $P||Q$. *Then* $P||Q \preceq_s P'||Q'$.

5 Conclusion

In this paper, we have proposed a semantics for assume-guarantee specifications of the form $\rho(A) \triangleright \rho'(G)$ where ρ, ρ' are the least or greatest fixed point operators, and sound rules for composing such specifications. A salient feature of our semantics and these rules is that they provide a common formulation for both least and greatest fixed points and arbitrary combinations of them in assumption-guarantee specifications. Using these general rules, we have derived the first formulation of assume-guarantee rules for linear temporal logic formulas $\varphi \ \mathrm{U_W} \ \psi$, and a formulation of assume-guarantee rules for Moore machine with a trace-tree semantics.

The general rule also allows us to show that the previously proposed rules of [12,7] can be derived as special instances. It has been well-known that assume-guarantee rules are not propositionally valid and their soundness requires non-propositional reasoning in some guise. In our proofs of the results of [12,7], it is worth noting that once it was shown that the premises of their rules can be cast as satisfaction of assume-guarantee properties with respect to our semantics, the conclusions of their rules followed *only* by application of our rules (*Imp* $\triangleright$), (*Comp* $\nu \triangleright \nu$), and (*Sub* $\triangleright$) without requiring any non-propositional reasoning such as induction. In this sense, the three rules can be seen as a precise, uniform, and concise distillation of the non-propositional content of the soundness of any assume-guarantee reasoning principle. It is therefore our hope that a logic consisting of these three reasoning rules together with those of propositional logic would yield a powerful general logic for supporting circular compositional reasoning.

References

1. M. Abadi, and L. Lamport. Composing Specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, 1993.
2. M. Abadi, and L. Lamport. Conjoining Specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, 1995.
3. R. Alur, and T. A. Henzinger. Local liveness for compositional modeling of fair reactive systems. In *Proceedings of the Conference on Computer-Aided Verification*, pages 166–179, 1995.
4. R. Alur, and T. A. Henzinger. Reactive Modules. In *Proceedings of the IEEE Symposium on Logic in Computer Science*, pages 207–218, 1996.
5. A. Cau, and P. Collette. Parallel composition of assumption-commitment specifications: A unifying approach for shared variable and distributed message passing concurrency. *Acta Informatica*, 33:153–176, 1996.
6. O. Grumberg, and D. E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, 1994. Earlier version in *Proceedings of CONCUR 91: Concurrency Theory*, 1991.
7. T. A. Henzinger, S. Qadeer, S. K. Rajamani, and S. Tasiran. An assume-guarantee rule for checking simulation. In *FMCAD 98: Formal Methods in Computer-aided Design*, pages 421–432, 1998.
8. C. B. Jones. Tentative steps towards a development method for infering programs. *ACM Transactions on Programming Languages and Systems*, 5(4):596–619, 1983.
9. B. Jonsson, and Y. -K. Tsay. Assumption/guarantee specifications in linear-time temporal logic. *Theoretical Computer Science*, 167:47–72, 1996
10. R. P. Kurshan. *Computer-aided Verification of Coordinating Processes*. Princeton University Press, 1994.
11. K. McMillan. A compositional rule for hardware design refinement. In *Proceedings of the Conference on Computer-Aided Verification*, pages 24–35, 1997.
12. K. McMillan. Circular compositional reasoning about liveness. In *CHARME 99: Correct Hardware Design and Verification*, pages 342–345, 1999.
13. J. Misra, and K. M. Chandy. Proofs of network processes. *IEEE Transactions on Software Engineering*, SE-7(4):417–426, 1981.
14. P. K. Pandya, and M. Joseph. P-A logic — A compositional proof system for distributed programs. *Distributed Computing*, 5:37–54, 1991.
15. A. Pnueli. In transition from global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems*, pages 123–144, 1984.
16. E. W. Stark. A proof technique for rely-guarantee properties In *Proceedings of the Conference on the Foundations of Software Technology and Theoretical Computer Science*, pages 369–391, 1985.
17. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

A PTAS for Minimizing Weighted Completion Time on Uniformly Related Machines
(Extended Abstract)

Chandra Chekuri[1] and Sanjeev Khanna[2]

[1] Bell Labs, 600 Mountain Ave, Murray Hill, NJ 07974.
chekuri@research.bell-labs.com.
[2] Dept. of CIS, University of Pennsylvania, Philadelphia, PA 19104.
sanjeev@cis.upenn.edu. Supported in part by an
Alfred P. Sloan Research Fellowship.

Abstract. We consider the well known problem of scheduling jobs with release dates to minimize their average weighted completion time. When multiple machines are available, the machine environment may range from identical machines (the processing time required by a job is invariant across the machines) at one end of the spectrum to unrelated machines (the processing time required by a job on each machine is specified by an arbitrary vector) at the other end. While the problem is strongly NP-hard even in the case of a single machine, constant factor approximation algorithms are known for even the most general machine environment of unrelated machines. Recently a PTAS was discovered for the case of identical parallel machines [1]. In contrast, the problem is MAX SNP-hard for unrelated machines [11]. An important open problem was to determine the approximability of the intermediate case of uniformly related machines where each machine has a speed and it takes p/s time to process a job of size p on a machine with speed s. We resolve the complexity of this problem by obtaining a PTAS. This improves the earlier known approximation ratio of $(2 + \epsilon)$.

Keywords: Polynomial time approximation scheme; average completion time; scheduling; uniformly related machines; weighted completion time.

1 Introduction

Scheduling to minimize average weighted completion time is one of the most well studied class of problems in scheduling theory. In this paper we concentrate on the following variant. We are given a set of n jobs where each job j has a processing time p_j, a weight w_j, and a release date r_j before which it cannot be scheduled. The goal is to schedule the jobs on a set of m machines *non-preemptively* with the objective of minimizing $\sum_j w_j C_j$ where C_j is the completion time of j in the schedule. The specific machine environment we consider in this paper is the *uniformly related* case. Machine i has a *speed* s_i and it takes p_j/s_i time for machine i to process job j. In the $\alpha \,|\, \beta \,|\, \gamma$ scheduling notation introduced by Graham et al. [7] this problem is denoted by $Q|r_j|\sum_j w_j C_j$. Using

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 848–861, 2001.

some non-trivial extensions to the ideas introduced in [1] we obtain a polynomial time approximation scheme (PTAS) for this problem. Our ideas also extend to the preemptive case $Q|r_j, pmtn|\sum_j w_j C_j$ but we omit the details of that result in this extended abstract.

Most variants of scheduling to minimize average completion time are strongly NP-hard including preemptive problems [12]. Polynomial time solvable cases include $P||\sum_j C_j$, $1||\sum_j w_j C_j$, and $R||\sum_j C_j$. In the last few years considerable progress has been made in understanding the approximability of many of these NP-hard problems. Constant and logarithmic ratio approximations were found for several variants: diverse machine environments (one, parallel, unrelated) and with a variety of constraints on the jobs (release dates, precedence constraints, delays) [14,8,3,15,6,13]. See [8,1] for more details on the history of these developments. Hoogeven et al. [11] obtained MAX SNP-hardness for some problems, especially those that had precedence constraints on jobs and/or involved unrelated machines. These results led to some conjectures regarding the approximability of variants with only release dates on the jobs. In particular, the problem $1|r_j|\sum_j w_j C_j$ was conjectured to have a PTAS, and $P|r_j|\sum_j w_j C_j$ was conjectured not to have a PTAS (the problem $R|r_j|\sum_j W_j C_j$ was shown to be MAX SNP-hard in [11]). Most of the ideas that led to constant factor approximation algorithms did not seem to lead to the design of approximation schemes. They were based on either preemptive relaxations or linear programming relaxations that had integrality gaps. Skutella and Woeginger [16] obtained a PTAS for the problem $P||\sum_j w_j C_j$ using some ideas from Alon et al. [2]. The basic idea in [16] was to group jobs based on similar values of w_j/p_j and then finding good schedules for each group separately. The schedules for the different groups could be combined together on the same machine using Smith's rule since there are no release dates. These ideas do not have a straight forward extension when jobs have release dates, especially in a multiple machine environment. More recently substantial progress was made in [1] where PTASs were obtained for scheduling jobs with release dates on single, identical parallel machines, and a constant number of unrelated machines both with and without preemptions allowed.

The above mentioned results improved our understanding of the approximability of scheduling with release dates by showing that problems admitting a PTAS (identical parallel and constant number of unrelated) were sufficiently close to the case that is MAX SNP-hard (unrelated machines). An open problem that remained was to determine the approximability of the related machine case, a strong generalization of the identical machine problem, and an important special case of the unrelated machine problem. In this paper we obtain a PTAS for this case, improving the earlier known ratio of $(2+\epsilon)$ [15].

Techniques and Relation to Previous Work: Scheduling on related machines generalizes the case of identical parallel machines in a natural way. Not surprisingly, we use as our starting point a dynamic programming framework presented in [1]. Informally speaking, the framework requires three key problems to be solved: (i) how to maintain polynomial size description of jobs that remain to be scheduled at any given time, (ii) a polynomial size description of how machines interact as one proceeds from one instant of time to the next, and (iii) a polynomial-

time algorithm for $(1+\epsilon)$-approximating the special case where we have only a constant number of distinct release dates. Unfortunately, the ideas used for solving these problems in the identical machines case do not generalize easily to the case of related machines. The main contribution of this paper is the development of the substantial new technical machinery required to obtain a PTAS for our problem. In what follows we give a brief overview of the difficulties involved in moving from the identical machine to the related machine case. In the identical machine case we could separate jobs into large and small based on their size. A critical part of the earlier approximation schemes was that at any point in time there are only $O(1)$ distinct large job sizes to consider. This allowed for explicit maintenance of certain parameters associated with each large job size class (such as how many are left, how many to schedule etc.) in time and space $m^{O(1)}$. The small jobs are easy to handle using a greedy approach. At a high level the main difference in the related machine case is in the possibility of having up to $\Omega(\log m)$ geometrically spaced speeds (we show how to reduce an arbitrary instance to such a restricted instance). The fastest machine could be m times faster than the slowest one. Because of the different speeds, jobs cannot be classified as large and small in an absolute manner. Thus, at any time instant, there could be up to $\Omega(\log m)$ job sizes that could potentially be executed as large. The dynamic programming framework of [1] gives a running time of $m^{\Theta(\log m)}$. Each of the three key steps in the framework has this dependence on the number of speeds.

To get around these difficulties we use two approaches. First, we show the existence of approximate schedules with *weaker* requirements than previously known. In particular, we give a procedure to preprocess the input in such a way that there is an approximate schedule in which every job finishes by $O(1)$ times its release date.

Second, we use more sophisticated enumeration and dynamic programming techniques. We mention two main ideas in this context. We use an approximate guessing tool from the recent work on the multiple knapsack problem [4] that allows, under certain conditions, to reduce an enumeration that would take $\Omega(m^{\log m})$ time if done naively to $O(\text{poly}(m))$ time. One of the components in [1] is an approximation scheme for the special case where the number of release dates is a fixed constant. This component is substantially more complicated in the related machines case. In this regard, the differences are akin to the differences in the approximation schemes for minimizing makespan (without release dates) on identical machines [9] and on related machines [10]. In particular, we use ideas from [10] which requires dynamic programming across machines with different speeds.

In this extended abstract we concentrate on getting the central ideas across and we omit formal proofs and several technical details in the interest of clarity and conciseness. We focus here only on the non-preemptive case. Various dependencies on ϵ have not been optimized.

2 Preliminaries

We first discuss some general techniques and lemmas that apply throughout our paper. We aim to transform any input into one with simple structure. Some of these transformations will be similar to those in [1] but some are new to the related machine case. A key shifting transformation is considerably more involved in its implementation and proof of correctness. We sequence several transformations of the input problem. Each transformation potentially increases the objective function value by $1+O(\epsilon)$, so we can perform a constant number of them while still staying within $1+O(\epsilon)$ of the original optimum. Using notation consistent with [1] we say that a transformation produces $1+O(\epsilon)$ *loss*. To argue that a transformation does not produce more than a $1+O(\epsilon)$ loss we typically take an optimal schedule and show how a near optimal schedule exists with the properties desired after the transformation. We go over two such ideas since we will be using them repeatedly. The first is ordering certain subsets of jobs by the ratio w_j/p_j (Smith's ratio). This is motivated by Smith's optimal algorithm for scheduling on a single machine when jobs are all released at the same time [17]. When we have many jobs that are released at the same time we will be able to show that there are approximate schedules that use this order in selecting the jobs for execution. The second transformation is *time stretching*. It is best understood by mapping time t to $(1+\epsilon)t$. Consider a single machine schedule where we map the completion time of each job according to the above mapping. This will worsen the schedule value by only a $(1+\epsilon)$ factor. However, since the processing times of the jobs remain the same, this leaves extra "space" in the schedule which we exploit to schedule other jobs. This allows us to obtain schedules with nicer structure while losing only a $1+O(\epsilon)$ factor. After the preprocessing of the input we use a dynamic programming framework to find a schedule with a special structure that is guaranteed to be within a $1+O(\epsilon)$ factor of the optimum.

Notation: For simplicity we will assume throughout the paper that $1/\epsilon$ is integral. We use C_j and S_j to denote the completion time and start time respectively of job j, and OPT to denote the weighted completion time of an optimal schedule. The number of jobs and machines is denoted by n and m respectively. We denote the speed of a machine i by s_i and assume w.l.o.g. that $s_1 \geq s_2 \geq \ldots \geq s_m$.

2.1 Input Transformations

We start with some transformations that are generalizations of those in [1]. We then introduce several new ones that are crucial for the related machines case.

Geometric Rounding: Our first simplification creates a well-structured set of possible processing times, release dates, and machine speeds.

Lemma 1. *With* $(1+\epsilon)$ *loss, we can assume that processing times, release dates, and machine speeds are integer powers of* $1+\epsilon$.

For an arbitrary integer x, we define $R_x := (1+\epsilon)^x$. As a result of Lemma 1 we can assume that all release dates are of the form R_x for some integer x.

We partition the time interval $(0, \infty)$ into disjoint intervals of the form $I_x := [R_x, R_{x+1})$. We will use I_x to refer to both the interval and the size $(R_{x+1} - R_x)$ of the interval. We will often use the fact that $I_x = \epsilon R_x$, i. e., the length of an interval is ϵ times its start time. Observe that the notion of time is independent of the machine speeds.

Large and Small Jobs: As in [1] we classify jobs as small and large. Jobs are small if their processing time is sufficiently small relative to the interval in which they run. Large jobs are those that take up a substantial portion of the interval. Note that this definition is both a function of the job size and the interval. A difficulty with related machines is that a job in an interval could be small or large depending on the machine on which it is processed. Therefore we say that a job is large or small by qualifying with the speed we have in mind. To be more precise we say that a job p_j is *small* with respect to an interval I_x for speed s_ℓ if $p_j/s_\ell \leq \epsilon^3 I_x$, otherwise it is *large*. We will often simply say that a job p_j is scheduled as small to indicate that it will be scheduled in some interval I_x on some machine with speed s_ℓ so that $p_j/s_\ell \leq \epsilon^3 I_x$. Similarly for large jobs. The following lemma states that a job is not arbitrarily large relative to its start time.

Lemma 2. *With $1+\epsilon$ loss, we can enforce $S_j \geq \epsilon p_j / s_{k(j)}$ for all jobs j where $k(j)$ is the machine on which j is processed.*

Crossing Jobs: While most jobs run completely inside one interval, some jobs *cross* over multiple intervals, creating complexity we would like to avoid. The next two lemmas simplify this problem: we can assume that no job crosses too many intervals, and we can assume there are no small crossing jobs at all.

Lemma 3. *With $1+\epsilon$ loss we can ensure that every job crosses at most $s := \lceil \log_{1+\epsilon}(1 + \frac{1}{\epsilon}) \rceil$ intervals.*

Lemma 4. *With $1+\epsilon$ loss we can restrict attention to schedules in which no small job crosses an interval.*

Lemma 5. *With $1+\epsilon$ loss we can restrict attention to schedules in which each job that is scheduled as large starts at one of $1/\epsilon^4$ equi-spaced instants within any interval.*

***O*(log *m*) Speed Classes.** It is easy to see that rounding the speeds to powers of $(1+\epsilon)$ results only in an $1+O(\epsilon)$ loss. However, this can still leave m distinct speeds. We reduce the number of distinct speeds to $O(\log m)$ as follows. Intuition suggests that machines much slower than the fastest machine can be ignored, with little loss in the schedule value. We formalize this intuition below. Let $s_1 \geq s_2 \geq \ldots s_m$ be the speeds of the machines. We can assume that $m > 1/\epsilon^3$ for otherwise we can use the algorithm in [1] to obtain a PTAS.

Lemma 6. *With $(1+\epsilon)$ loss we can ignore machines with speed less than $\frac{\epsilon}{m} s_{1/\epsilon^3}$.*

Proof Sketch. Consider an optimal schedule S and let $k(j)$ be the machine on which job j is executed in S. Let $A_i = \sum_{k(j)=i} w_j C_j$ be the contribution of machine i to the schedule value. Let ℓ be such that $A_\ell = \min_{2 \le i \le 1/\epsilon^3} A_i$. We obtain a new schedule as follows. We remove the jobs allocated to M_ℓ and execute them on M_1 in a delayed fashion. By time stretching on M_1 it is clear that we can execute the jobs of M_ℓ with no more than a $1/\epsilon^2$ factor delay. Since $A_\ell \le \epsilon^3 \cdot \text{OPT}$ this does not effect the schedule by more than a $(1+\epsilon)$ factor. We schedule the jobs allocated on all the slow machines (the ones with speed smaller than $s_{1/\epsilon^3} \cdot \frac{\epsilon}{m}$) and assign them to M_ℓ. We do this as follows. All the jobs that start in each of the slow machines in the interval I_x are scheduled in the interval I_x on M_ℓ. It is easy to see that all the jobs will complete on M_ℓ within the same interval I_x and hence their completion times are affected by no more than a $(1+\epsilon)$ factor.

Following Lemma 1 and Lemma 6 we can assume that our instance has $O(\log m/\epsilon)$ distinct speeds. We group machines with the same speed and refer to the group as a speed class. Let K be the exact number of classes we have with the implicit understanding that $K = O(\log m)$. For $1 \le i \le K$, let M_i denote the machines in the ith speed class. Let $m_i = |M_i|$ denote the number of machines in M_i and let s_i denote the common speed of machines in M_i. We assume w.l.o.g. that $s_1 > s_2 > \ldots > s_K$.

Generating Extra Machines. We describe another technique that allows us to simplify things. The lemma below shows that any schedule can be transformed into a $1 + O(\epsilon)$-approximate schedule where we use only a $(1-\epsilon)$-fraction of the machines from any sufficiently large machine class. Note that a similar lemma does not hold for minimizing makespan. We will assume from here on that we are working with this reduced allocation of machines. The remaining extra machines would be useful in a key step for implementing the dynamic programming.

Lemma 7. *Given m machine instance of identical parallel machines where $m > 1/\epsilon^3$ there is a $1 + O(\epsilon)$-approximate schedule on $m(1-\epsilon^3)$ machines.*

The proof of the above lemma uses a similar line of reasoning as in the proof of Lemma 6.

Shifting. Our next goal is to show that we can preprocess the input instance I in such a way that we can guarantee a schedule in which every job will be completed within a constant number of intervals from its release. We accomplish this by selectively retaining only a fraction of the jobs released in each interval and *shifting* the rest to later intervals. This basic idea plays a crucial role in obtaining the PTAS for the parallel machine case $P|r_j|\sum_j w_j C_J$ [1]. A brief description of the procedure in [1] follows. Jobs released in an interval I_x are either small or fall in to one of $O(1)$ large size classes. Small jobs are ordered in non-increasing order according to the ratio w_j/p_j and large jobs are ordered in each size class in decreasing order of their weights. In each class the number of jobs retained is restricted by the volume that could be processed in the interval I_x. The rest are shifted to the next interval. Since the number of classes is $O(1)$

the total volume of jobs released at R_x in the modified instance is $O(1)$ times the volume of I_x. By time shifting one can show that there exists an approximate schedule in which all the jobs at I_x could be finished within $O(1)$ intervals after R_x. This enables locality in dynamic programming.

There is no simple generalization of the above ideas to the related machine case because the notion of small and large jobs is now relative to the machine speed. The number of distinct job sizes that can be executed as large in an interval could be $\Omega(\log m)$ and we cannot afford to have a volume of jobs released at I_x that is $\Omega(\log m)$ times the processing capability of the machines in I_x. We design a new procedure below that essentially retains the property concerning the volume. The proof that this procedure results in only an $1 + O(\epsilon)$ loss is however more involved. We describe the shifting procedure formally below.

Let J_x be the set of jobs released at R_x. For each speed class i from K down to 1 the following process is done.

- Let T_x^i and H_x^i be the small and large jobs with respect to speed s_i released at R_x that are still to be processed.
- The number of distinct size classes in H_x^i is $O(1/\epsilon^2)$. In each size class we pick jobs in order of non-increasing weights until the sum of processing times of jobs picked just exceeds $m_i s_i I_x/\epsilon^2$ or we run out of jobs.
- We pick jobs in T_x^i in non-increasing w_j/p_j ratio until the processing time of the jobs picked just exceeds $m_i s_i I_x$ or we run out of jobs.
- We remove the jobs picked from T_x^i and H_x^i from J_x.

Jobs that are not picked in any speed class are shifted to the next interval. We repeat this process with each successive interval. Let I' be the modified instance obtained after the shifting process above and for each x let J'_x be the set of jobs released at R_x in I'.

Lemma 8. *For any given rounded job size s let $a_x^s(S)$ and $b_x^s(S)$ denote the number of jobs of size s started in I_x as small and large respectively in an optimal schedule S. There exists a $1 + O(\epsilon)$-approximate schedule S' such that for each s and x either $a_x^s(S') < \frac{1}{\epsilon^2} b_x^s(S')$ or $b_x^s(S') = 0$.*

Proof. Consider an optimal schedule S. Suppose $a_x^s(S) > \frac{1}{\epsilon^2} b_x^s(S)$ for some size s and interval x. We create a modified schedule S' as follows. We take all the $b_x^s(S)$ jobs executed as large and execute them as small within the same interval I_x by scheduling them on faster machines. Since the number of jobs executed as small, $a_x^s(S)$, is much larger that those executed as large, we claim that this can be accomplished by stretching the interval by only a $(1 + \epsilon)$ factor. In the modified schedule $b_x^s(S') = 0$. This can be done simultaneously for all s and x which do not satisfy the lemma and no interval stretches by more than a $1 + \epsilon$ factor. The schedule S' is a $1 + O(\epsilon)$-approximation to S.

Lemma 9. *For the modified instance I' obtained from I by the shifting procedure*

1. $\text{OPT}(I') \leq (1 + O(\epsilon))\text{OPT}(I)$.
2. *There exists a $(1 + O(\epsilon))$-approximate schedule for I' in which all jobs in J_x are finished by $R_{x+O(\log(1/\epsilon)/\epsilon)}$.*

Proof Sketch. We prove (2) first. Let J_x^i be the set of jobs picked by the shifting procedure in speed class i, $1 \leq i \leq K$ at R_x. We note that all jobs in J_x^i can be executed by machines of speed class i in time $O(I_x)$. This implies that $p(J_x^i)$ will be small relative to interval $I_{x+O(\log(1/\epsilon)/\epsilon)}$ because of the geometrically increasing property of interval sizes. Therefore time stretching any arbitrary but fixed optimal schedule allows us to create the required space to execute all the jobs in J_x^i by then.

Now we prove (1). We observe that the shifting procedure does the following. For each size class s that can be executed as large in I_x the procedure picks the n_x^s/ϵ^2 jobs in non-increasing weight order from J_x where n_x^s is the maximum number of jobs that can executed as large of size class s in I_x. From Lemma 8 there exists a $(1 + O(\epsilon))$-approximate schedule in which the jobs executed as large in I_x of size s are contained in the set we pick. The small jobs that are executed in I_x can be treated as fractional jobs and this enables us to pick them in a greedy fashion in non-increasing order of w_j/p_j and we pick enough jobs to fill up the volume of I_x. The proof of the near optimality of greedily picking small jobs is similar to that of the parallel machine case in [1] and we omit the details in this version.

2.2 Overview of Dynamic Programming Framework

We give a brief overview of the dynamic programming framework from [1].

The idea is to divide the time horizon into a sequence of blocks, say $\mathcal{B}_1, \mathcal{B}_2, ...$, each containing a constant number (depending on ϵ) of intervals dates, and then do a dynamic programming over these blocks by treating each block as a unit. There is interaction between blocks since jobs from an earlier block can cross into the current block. We choose the number of intervals in each block to be sufficiently large so that no job crosses an entire block. From Lemma 3 we conclude that $O(1/\epsilon^2)$ intervals per block suffice. In other words jobs that start in $\mathcal{B}_i$ finishes no later than $\mathcal{B}_{i+1}$. A *frontier* describes the potential ways that jobs in one block finish in the next. An incoming frontier for a block $\mathcal{B}_i$ specifies for each machine the time at which the crossing job from $\mathcal{B}_{i-1}$ finishes on that machine. Let $\mathcal{F}$ denote the possible set of frontiers between blocks. The dynamic programming table maintains entries of the form $O(i, F, U)$: the minimum weighted completion time achievable by starting the set U of jobs in block $\mathcal{B}_i$ while leaving a frontier of $F \in \mathcal{F}$ for block $\mathcal{B}_{i+1}$. Given all the table entries for some i, the values for $i + 1$ can be computed as follows. Let $C(i, F_1, F_2, V)$ be the minimum weighted completion time achievable by scheduling the set of jobs V in block $\mathcal{B}_i$, with F_1 as the incoming frontier from block $\mathcal{B}_{i-1}$ and F_2 as the outgoing frontier to block $\mathcal{B}_{i+1}$. We obtain the following equation.

$$O(i+1, F, U) = \min_{F' \in \mathcal{F}, V \subset U} \big(O(i, F', V) + C(i+1, F', F, U - V) \big)$$

3 Implementing the Dynamic Programming Framework

Broadly speaking, we need to solve three problems for using the dynamic programming framework described in the preceding section. First, we need a mechanism to compactly describing for any block $\mathcal{B}_i$, the set of jobs that were released prior to $\mathcal{B}_i$ and have already been scheduled. Second, we need to ensure that the number of distinct frontiers in $\mathcal{F}$ is polynomial for any block $\mathcal{B}_i$. Finally, given a set of jobs to be scheduled within a block, we should be able to find a $(1+\epsilon)$-approximate schedule. A basic theme underlying our implementation of these steps is to relax the requirements slightly. In the parallel machine case we could enumerate the set of jobs U that are released in $\mathcal{B}_i$ and started in $\mathcal{B}_i$ itself. This was done by separating out the small and large jobs. Since there were only $O(1)$ large job sizes in each $\mathcal{B}_i$ this was relatively easy. Now we have $\Omega(K)$ large job sizes. We would have to enumerate $m^{\Omega(K)}$ possibilities. To get around this difficulty we use a global scheme that is inspired by the recent work on the multiple knapsack problem [4]. We will be able to figure out most of the important jobs using the above scheme in polynomial time and we show this approximate enumeration suffices. A similar situation arises in enumerating the frontiers. Here we use a different idea based on Lemma 7. Finally, another difficult part is the problem of scheduling jobs in a fixed number of intervals. The approach we adopt is some what akin to the approach taken by Hochbaum and Shmoys [10] to obtain a PTAS for the makespan problem on related machines. The basic idea is to do dynamic programming across the speed classes going from the slowest speed class to the fastest. The advantage of this approach is the following. Any fixed size class is large for only $O(1)$ consecutive speed classes because of the geometrically increasing speeds. This implies that while we are doing the dynamic programming the number of size classes for which we have to maintain detailed information (in terms of the exact number of jobs remaining etc) is only $O(1)$ as opposed to $\Omega(K)$ if we tried to solve the problem all at once. The many subtle details that we need to make all these ideas work are explained in the remainder of this section.

In what follows, we assume each block consists of $\alpha = O(1/\epsilon^2)$ intervals, the precise constant is of not much importance.

3.1 Compact Description of Remaining Jobs

We start by observing that by Lemma 9 and our choice of block size, there exists a $(1+\epsilon)$-approximate schedule such that all jobs released in a block $\mathcal{B}_i$ are always scheduled by the end of the block $\mathcal{B}_{i+1}$. In fact we will be able to schedule all jobs released in $\mathcal{B}_i$ by the end of $\mathcal{B}_{i+1}$ irrespective of how many of them have been executed in $\mathcal{B}_i$ itself. We will restrict our attention to only such schedules. Thus, to compactly describe the set of jobs that remain from B_i, we need only describe a mechanism for compact representation of the set of jobs chosen to be scheduled within B_i. However, due to the non-uniform nature of machine speeds, this process turns out to be more involved than the identical machine case. In particular, we rely on some ideas from the recent approximation scheme for the multiple knapsack problem [4]. We show that there exists a $(1+\epsilon)$-approximate

schedule that needs to enumerate over only a polynomial number of distinct possibilities for sets of jobs chosen for scheduling within a block $\mathcal{B}_i$. We will use the following elementary fact from [4]:

Proposition 1. *Let $h = O(\log m)$. Then the number of h-tuples $\langle k_1, k_2, ..., k_h \rangle$ such that $k_i \in [0...h]$ and $\sum k_i \leq h$ is $m^{O(1)}$.*

We can now describe our scheme for enumerating the job subsets. For each interval $I_j \in \mathcal{B}_i$ we separately enumerate the jobs that are released at I_j and will be scheduled in $\mathcal{B}_i$. Since the number of intervals in each block is a constant depending only on ϵ, we concentrate on a single interval. Let X_l be the set of all jobs released in I_j that are large for the slowest speed and let X_s be the remaining jobs. We focus here on the enumeration of the set X_l and later sketch the idea for the set X_s. Let w be the total weight of jobs in X_l. As a result of our shifting procedure, total number of jobs in X_l can be bounded by $m^2 f(1/\epsilon)$ for some suitably large function f. We use this fact to ignore from our consideration all jobs in X_l whose weight is less than $\delta(\epsilon) \cdot w/m^2$ where δ is a suitably smaller than $1/f(1/\epsilon)$ — we will schedule these jobs in $\mathcal{B}_{i+1}$ and by our choice of δ, their completion time can be amortized to an ϵ fraction of the weighted completion of other jobs in X_l. Once we eliminate these jobs from consideration, there are only $O(\log m)$ distinct weights for the remaining jobs in X_l.

1. For interval $I_j \in \mathcal{B}_i$ we first specify W_j the total weight of jobs in X_l that will be scheduled in $\mathcal{B}_i$. We specify this weight in multiples of $\delta(\epsilon) \cdot w/m^2$ by an integer ℓ such that $0 \leq \ell \leq m^2/\delta(\epsilon)$. The set of jobs that are lost due to the downward rounding are scheduled in $\mathcal{B}_{i+1}$ and as above the increase in the schedule value can be bounded. The number of choices for ℓ is polynomial in m.
2. For a given W_j (specified by the integer ℓ), we specify a partition of W_j into $h = O(\log m)$ classes one for each of the distinct large size classes. Since an exact partition would require quasi-polynomial possibilities, we need to do this step approximately. We specify an approximation to an exact partition of the form $\langle W_j^1, W_j^2, ..., W_j^h \rangle$ by guessing an integer vector $\langle k_1, k_2, ..., k_h \rangle$ such that $k_l(\delta(\epsilon) \cdot W_j/h) \leq W_j^l < (k_l + 1)(\delta(\epsilon) \cdot W_j/h)$. By Proposition 1, the number of tuples enumerated above is bounded by $m^{O(1)}$ for any fixed $\epsilon > 0$. The error introduced by this approximate under-guessing can be bounded by $\delta(\epsilon) \cdot W_j$ over all h size classes. Since all jobs released in $\mathcal{B}_i$ are always scheduled by the end of the block $\mathcal{B}_{i+1}$, the cost of the schedule as a result of the under-guessing above increases by at most a factor of $1 + O(\epsilon)$.
3. Finally, for each size class of jobs released in I_j, we greedily pick the smallest number of jobs whose cumulative weight exceeds the weight guessed above.

For jobs in X_s, we order them by w_j/p_j and guess the fraction of them that will be scheduled with a precision of $\delta(\epsilon)/m^2$, and using similar reasoning as above, conclude that we do not incur more than a $1 + \epsilon$ loss.

In summary we showed that by restricting the choice to important jobs based on weights we need to consider only a polynomial number of sets as candidates for jobs scheduled within a block.

3.2 Frontiers

A frontier describes the set of jobs that are crossing over from a block $\mathcal{B}_i$ to the next block $\mathcal{B}_{i+1}$. By Lemma 4, we know that only a job that is scheduled as large can participate in a frontier, and by Lemma 5 we know that there are only $1/\epsilon^4$ distinct time instants in any interval by which a job scheduled as large starts or ends. Further the number of distinct job sizes that can execute as large in a block is $O(1/\epsilon^4)$. Hence a crossing job on a large machine can be specified by the size and the time instant it starts in the interval. Let $q = O(1/\epsilon^8)$ denote the total number of such distinct frontiers for any machine. In order to describe the over all frontier, we need to specify this information for each machine. Consequently, we can describe the frontier by a vector $\langle m_{11}, m_{12}, ..., m_{1q}, m_{21}, ..., m_{Kq}\rangle$ where m_{ij} denotes the number of machines in the speed class $\mathcal{C}_i$ that have a job finishing at the jth distinct instant in block $\mathcal{B}_{i+1}$. Clearly, an exact enumeration would require considering quasi-polynomial number of possibilities. We now argue that in order to obtain a $(1+\epsilon)$-approximation it suffices to work with a polynomial-sized set $\mathcal{F}$ of frontiers. With any vector of the above form, we associate a vector $\langle l_{11}, l_{12}, ..., l_{1q}, l_{21}, ..., l_{Kq}\rangle$ in $\mathcal{F}$ where $l_{ij} = m_{ij}$ if $m_i \leq 1/\epsilon^3$, and otherwise, $(l_{ij} - 1)(\epsilon^{11} m_i) < m_{ij} < l_{ij}(\epsilon^{11} m_i)$. Clearly, there are only $O(1/\epsilon^{11K}) = m^{O(1)}$ such vectors to be considered. However, the above approximation of an exact frontier description *over-allocates* machines for large machine classes, and thus would necessitate extra machines. The total number of extra machines needed by any large speed class is bounded by $\epsilon^{11} \cdot q \cdot m_i$ which is at most $\epsilon^3 \cdot m_i$. We allocate these extra machines by using Lemma 7 which allowed us to keep aside an $\epsilon^3 \cdot m_i$ machines for each speed class of size at least $1/\epsilon^3$.

3.3 Scheduling Jobs within a Block

We now describe a $(1 + \epsilon)$-approximate implementation of the procedure $C(i, F_1, F_2, Z)$. Recall that $C(i, F_1, F_2, Z)$ is the procedure that computes the best schedule for a set of jobs Z that are to be scheduled in block $\mathcal{B}_i$ with incoming and outgoing frontiers specified by F_1 and F_2.

In what follows, it will be useful to assume that F_1^j and F_2^j denote the components of F_1 and F_2 that correspond to the jth speed class M_j. Our scheduling procedure is based on a dynamic programming across the classes; we move from the slowest class to the fastest class and treat each class as a unit. In [1] a procedure was given to schedule on a single speed class. The basic idea was to enumerate large job placements and schedule the small jobs greedily in the space left by the large jobs. Enumerating the large job placements was relatively easy because there were only $O(1)$ sizes that were large in each block. We do not know how to efficiently enumerating all large job placements with K speed classes in polynomial time, hence we resort to doing dynamic programming across classes. When considering a current class we would like to know the jobs that are already scheduled in the preceding classes. The main difficulty in implementing the dynamic program is to maintain a compact description of this information. To achieve this we use the notion of *template schedules*.

Template Schedules: A template schedule at a machine class M_j provides information about the jobs that remain to be scheduled along with some "coarse" information constraining how these jobs may be scheduled. It is this additional scheduling information that implicitly encodes information concerning weights of the remaining jobs. Specifically, a template schedule at a machine class M_j specifies scheduling information for all jobs that are eligible to be scheduled as large on a machine in M_j, as well as global information concerning the volume of jobs that must be scheduled as small on machines in M_{j-1} through M_1. We describe these two aspects next.

Let $L(j)$ denote the set of job sizes that can be scheduled as large at the jth machine class, and let $Z_{L(j)}$ denote the job set Z restricted to the sizes in $L(j)$. At the jth speed class we consider all possible extensions of template schedules for the $(j-1)$th machine class so as to incorporate scheduling information for the jobs in the set $Q = Z_{L(j)} \setminus Z_{L(j-1)}$. A template schedule specifies the following information for each size class in Q.

- The number of jobs that are executed as large and the number that are executed as small.
- For those executed as small, the number that will be executed in each interval of $\mathcal{B}_i$.
- For those executed as large, the number that will be executed for each possible placement in each of the speed classes where that size can be executed as large. We note that this information includes speed classes greater than j, that is classes that have already been processed.

Lemma 10. *The template schedule information is polynomial size for each size class in the set Q.*

Proof Sketch. For any fixed size class the number of speed classes on which it can be scheduled as large is $O(\log(1/\epsilon)/\epsilon)$ since the speeds are increasing geometrically. Further the number of distinct start times of large jobs in each class is also fixed for fixed ϵ, following Lemma 5. Hence specifying the numbers is polynomial.

At each class M_j the number of job sizes in Q is $O(1/\epsilon^2)$, hence the total information for all sizes in Q is still polynomial. Observe that template schedules do not maintain any explicit information about the weight of the jobs that remain. However, this information is implicit and can be recovered as follows. Consider the scheduling at a machine class M_j that receives the template schedules for all job sizes that can be scheduled as large on a machine in M_j. Fix one such size class, say p_k, and let a_t denote the number of jobs of size p_k that are required to start at the tth starting time in block $\mathcal{B}_i$ on a machine in M_j. Since a template schedule completely determines the finishing times of all jobs of size p_k, it is straightforward to determine the weights associated with each one of the a_t jobs (we resolve all ties in some fixed canonical manner).

The idea of template schedule as described so far allows us to identify the jobs that are to be scheduled as large at any machine class. However, we need additional information to determine what jobs are available to be scheduled as small at any machine class. We do this by maintaining a vector of the form

$\langle V_1, ..., V_\alpha \rangle$ such that V_l specifies the total volume of the small jobs that must be scheduled in the lth interval of the block $\mathcal{B}_i$ in classes M_{j-1} through M_1.

Lemma 11. *The template schedule information for small jobs is of poly size.*

Proof Sketch. We claim that the precision needed for each V_i is $O(\epsilon^5/m^2)$. Assume without loss of generality that $s_K = 1$ and hence $s_1 = O(m)$. Consider the smallest large job in the block and let s be its size. From our assumption that $s_K = 1$, s is at least ϵ^3 times the smallest interval in $\mathcal{B}_i$. We claim that the volume can be maintained in multiples of ϵ^2 times s. This is because the size of each job in the block can be approximated to within a $(1+\epsilon)$ factor by multiples of the above quantity. Coupled with the fact that the total volume that can be executed in the block is $O(m^2)$ the lemma follows.

Dynamic Programming with Template Schedules. We maintain a table $T(j, X, Y)$ where j ranges over machine speed classes and X and Y are template schedules for M_j and M_{j-1} respectively. $T(j, X, Y)$ stores the best weighted completion time that is consistent with X and Y. Note that by knowing X and Y the job set that is to be scheduled in M_j is determined. Given X and Y computing $T(j, X, Y)$ involves the following.

- Checking for consistency between X and Y.
- Checking the feasibility of scheduling the jobs in M_j implied by X and Y.

Note that the template schedules implicitly determine the best weighted completion times. We briefly describe the feasibility computation below.

Scheduling Jobs within a Machine Class: For any machine class M_j, once we know the position of jobs to be scheduled as large, as well as the volume of jobs to be scheduled as small in each one of the α intervals, it is relatively easy to determine whether or not there exists a feasible schedule (with $1+\epsilon$ loss) that is consistent with this specification and the in-coming and out-going frontiers F_1^j and F_2^j.

Theorem 1. *There is a PTAS for the problem* $Q|r_j|\sum_j w_j C_j$.

References

1. F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation Schemes for Minimizing Average Weighted Completion Time with Release Dates. *FOCS '99.*
2. N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1:55–66, 1998.
3. S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. *ICALP '96.*
4. C. Chekuri and S. Khanna. A PTAS for the Multiple Knapsack Problem. *SODA '00.*
5. C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. *SODA '97.*

6. M. X. Goemans. Improved approximation algorithms for scheduling with release dates. *SODA '97*.
7. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.*, 5:287–326, 1979.
8. L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Offline and online algorithms. *Math. of OR*, 513–44, '97.
9. D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *JACM*, 34:144–162, 1987.
10. D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing*, 17:539–551, 1988.
11. J. A. Hoogeveen, P. Schuurman, and G. J. Woeginger. Non-approximability results for scheduling problems with minsum criteria. *IPCO '98*.
12. J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.
13. A. Munier, M. Queyranne, and A. S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. *IPCO '98*.
14. C. Phillips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming B*, 82:199–223, 1998.
15. A. S. Schulz and M. Skutella. Scheduling–LPs bear probabilities: Randomized approximations for min–sum criteria. *ESA '97*.
16. M. Skutella and G. J. Woeginger. A PTAS for minimizing the weighted sum of job completion times on parallel machines. *STOC '99*.
17. W. E. Smith. Various optimizers for single-stage production. *Naval Res. Logist. Quart.*, 3:59–66, 1956.

The Buffer Minimization Problem for Multiprocessor Scheduling with Conflicts

Marek Chrobak[1], János Csirik[2], Csanád Imreh[2], John Noga[3], Jiří Sgall[4], and Gerhard J. Woeginger[3]

[1] Dept. of Computer Science, University of California, Riverside, CA 92521, USA, marek@cs.ucr.edu
[2] József Attila University, Department of Computer Science, Árpád tér 2, H-6720 Szeged, Hungary, csirik,cimreh@inf.u-szeged.hu
[3] TU Graz, Institut für Mathematik B, Steyrergasse 30, A-8010 Graz, Austria, gwoegi,noga@opt.math.tu-graz.ac.at
[4] Mathematical Inst., AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic, sgall@math.cas.cz

Abstract. We consider the problem of scheduling a sequence of tasks in a multi-processor system with conflicts. Conflicting processors cannot process tasks at the same time. At certain times new *tasks* arrive in the system, where each task specifies the amount of work (processing time) added to each processor's workload. Each processor stores this workload in its *input buffer*. Our objective is to schedule task execution, obeying the conflict constraints, and minimizing the maximum buffer size of all processors. In the off-line case, we prove that, unless $P = NP$, the problem does not have a polynomial-time algorithm with a polynomial approximation ratio. In the on-line case, we provide the following results: (i) a competitive algorithm for general graphs, (ii) tight bounds on the competitive ratios for cliques and complete k-partite graphs, and (iii) a $(\Delta/2+1)$-competitive algorithm for trees, where Δ is the diameter. We also provide some results for small graphs with up to 4 vertices.

1 Introduction

We consider the problem of scheduling a sequence of tasks in a multi-processor system with conflicts. The term "conflict" refers to a situation where two or more processors share common resources that can only be accessed by one processor at any given time, e.g. specialized human operators, equipment (say, a printer or a phone line), materials, etc. Conflicting processors cannot process tasks at the same time. In other words, at any moment in time only a non-conflicting set of processors can be run simultaneously. At certain times new *tasks* arrive in the system, where each task specifies the amount of work (processing time) added to each processor's workload. Each processor stores this workload in its *input buffer*. Our objective is to schedule task execution, obeying the conflict constraints, and minimizing the maximum buffer size of all processors.

We model the multi-processor system as an undirected graph $G = (V, E)$, where V is the set of processors and the edges in E represent conflicts. The

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 862–874, 2001.

instance of the problem consists of a task sequence and a sequence of task arrival times. The sequence of task arrival times, denoted $\bar{t} = t^1, \ldots, t^m$, is a non-decreasing sequence of real numbers. The task that arrives at time t^k is given by a *task vector* $\boldsymbol{\tau}^k = (\tau_1^k, \ldots, \tau_n^k)$ of non-negative real numbers, and the whole task sequence $\boldsymbol{\tau}^1, ..., \boldsymbol{\tau}^m$ is denoted by $\bar{\boldsymbol{\tau}}$.

The state of the system at any time is fully described by the *load vector* $\boldsymbol{z} = (z_1, z_2, \ldots, z_n)$, where z_i denotes the amount of work stored in the input buffer of processor i. Upon the arrival of task $\boldsymbol{\tau}^k$, the state changes to $\boldsymbol{z} + \boldsymbol{\tau}^k$. In-between task arrivals, the processors execute the work stored in their buffers. At any time, an independent set of vertices I can be chosen for processing. If a processor i executes for σ time units during which no task arrives, it decreases its load from z_i to $\max\{z_i - \sigma, 0\}$.

A *schedule* is described by a sequence of independent sets I_j, $0 = 1, \ldots, m-1$ and an increasing sequence of times $\theta_0 = 0, \theta_1, \ldots, \theta_m$. During the time interval $[\theta_j, \theta_{j+1})$, the set of running processors is I_j. The load vector at any time is computed according to the rules above. A schedule is feasible for a given instance if $\theta_m > t^k$ and the load vector at time θ_m is the zero vector (i.e., all requests are processed). The *buffer size* of a schedule is the maximum of any coordinate in any load vector over all times. A schedule is called a *B-schedule* if it is feasible and its buffer size is at most B.

The buffer minimization problem can be now formulated as follows: given an instance $\langle G, \bar{t}, \bar{\boldsymbol{\tau}} \rangle$, where G is the conflict graph and $\bar{t}$, $\bar{\boldsymbol{\tau}}$ are the sequences of task arrival times and task vectors, find a B-schedule for $\langle G, \bar{t}, \bar{\boldsymbol{\tau}} \rangle$ with minimum buffer size B.

On-line buffer minimization. An on-line algorithm processes the workload from the buffers without knowledge of future tasks. In round k it learns the task vector $\boldsymbol{\tau}^k$ and determines a schedule that is identical with the current one at all times up to t^k and is feasible in case no further task arrives. This schedule is executed until the next task arrives, and then the process is repeated.

We evaluate on-line algorithms by comparing their buffer size to that of an optimal off-line algorithm. An on-line algorithm is *c-competitive* for a graph G if, for any task sequence that has a B-schedule, it constructs a feasible schedule with buffer size at most cB. The *competitive ratio of* G, denoted $\mathtt{buf}(G)$, is the infimum over all c for which there is a c-competitive on-line algorithm on G.

We also consider a slightly weaker definition, where the optimal size of buffers is fixed in advance. Without loss of generality, $B = 1$. We say that an on-line algorithm is *weakly c-competitive* for G if, for any task sequence that has a 1-schedule, it constructs a feasible schedule with buffer size at most c. The *weak competitive ratio of* G, denoted $\mathtt{buf}^-(G)$, is the infimum of all c for which there is a weak c-competitive on-line algorithm on G. Using the doubling technique to estimate the buffer size, it is quite easy to show that, for any graph G, the competitive ratio $\mathtt{buf}(G)$ is at most 4 times the weak competitive ratio $\mathtt{buf}^-(G)$.

Some algorithms are easier to describe if we allow them to process a convex combination of independent sets rather than a single set. For example, instead

of rapidly switching between two conflicting processors, we can run each of them at half speed. This generalization does not change the competitive ratio.

Our results. We introduce the buffer minimization problem as a model for studying various scheduling problems with conflicts. We believe that our model faithfully captures essential properties of scheduling multiprocessing systems over long periods of time. Other objectives in multiprocessor scheduling include fairness, load balancing and makespan minimization. Schedules that minimize the buffer size also typically perform well with respect to these other measures.

Off-line buffer minimization is closely related to fractional graph coloring. In fact, in Section 2, we show that it is even harder, namely that it has no polynomial time approximation algorithm with worst case guarantee that is polynomially bounded in the number n of processors, unless $P = NP$.

For the on-line case, we provide the following results:

(i) For every graph G, its competitive ratio $\texttt{buf}(G)$ is finite (Section 3).

(ii) The clique K_n has competitive ratio $\texttt{buf}(K_n) = \texttt{buf}^-(K_n) = H_n$ where $H_n = \sum_{i=1}^{n} 1/i$ is the nth harmonic number. For any complete bipartite graph $K_{m,n}$, $\texttt{buf}(K_{m,n}) = \texttt{buf}^-(K_{m,n}) = 2$. Further for complete k-partite graphs the competitive ratio is between H_k and $H_{k-1} + 1$ (Section 4). These upper bounds are achieved by a simple greedy algorithm.

(iii) For trees we show that their competitive ratio is at most $\Delta/2 + 1$, where Δ is the tree diameter (Section 5).

(iv) Finally, we provide bounds on the competitive ratios for graphs with up to four vertices (Section 6). All these ratios are between 1.5 and 2.5.

Previous work. The general concept of conflicts is not new in scheduling; in fact, one could argue that the whole area of scheduling is about resolving various types of conflicts for access to limited resources. In the classical literature on scheduling, conflicts are modeled by precedence relations between jobs and machine environments.

A model similar to ours was studied by Irani and Leung [3,12]. They also introduce a conflict graph, but in their work this graph represents conflicts between individual jobs (not processors), and the objective is to minimize makespan. They show that even for paths the competitive ratio is $\Omega(n)$. They also provide algorithms that are competitive for general graphs under some assumptions on the job arrival probabilities.

Similar problems to ours were also studied in relation to resource allocation in distributed systems, where the conflict graphs is often referred to as the *resource graph* [9]. For example, Bar-Noy et al [4] investigate resource allocation problems with the objective to minimize the average response time, and they provide some hardness and approximation results. The problems studied in [4] (see also [8] and the references in [4]) differ from ours in that they are *one-shot* resource allocation problems, while in our scenario we have a stream of tasks arriving over time. Our objective function is different as well. Finally, unlike in [4], where the allocation problems reduce to various color sum problems for graphs, the buffer minimization turns out to be closely related to fractional graph coloring.

Notation. States, or load vectors, will be denoted $\boldsymbol{z}$, $\boldsymbol{y}$, etc, possibly with superscripts. States of an on-line algorithm will be typically denoted by $\boldsymbol{a}$. $G = (V, E)$ denotes an undirected graph with n vertices. We will use the convention that $V = \{1, 2, \ldots, n\}$. By $N(v)$ we denote the set of neighbors of a vertex v in G. Given any non-negative vector $\boldsymbol{w} = (w_1, \ldots, w_n)$, and a vertex set $X \subseteq V$, we introduce the following notation:

$$\Sigma_X \boldsymbol{w} \;=\; \sum_{i \in X} w_i \quad \text{and} \quad \max_X \boldsymbol{w} \;=\; \max_{i \in X} w_i$$

For any real number x, we define $[x]^+ = \max\{0, x\}$, and for a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ let $[\boldsymbol{x}]^+ = ([x_1]^+, \ldots, [x_n]^+)$. For two n-dimensional vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, we write $\boldsymbol{x} \leq \boldsymbol{y}$ to denote that $\boldsymbol{x}$ is component-wise less than or equal to $\boldsymbol{y}$.

2 Buffer Minimization and Fractional Chromatic Number

We now exploit the relationship between the fractional chromatic number and the buffer size to show that the minimum buffer size is hard to approximate.

As usual, the chromatic number of a graph G is denoted by $\chi(G)$, and its clique number by $\omega(G)$. Consider a *weight* vector $\boldsymbol{w}$ indexed by the vertices of V, with $w_j \geq 0$ denoting the weight of vertex $j \in V$. The *weighted fractional chromatic number* $\chi_f(G, \boldsymbol{w})$ is the optimal objective value of the following linear program, where we use I to denote independent sets in G:

$$\begin{aligned} \chi_f(G, \boldsymbol{w}) \;&= \min \textstyle\sum_I x_I \\ &\text{s.t.} \;\; \textstyle\sum_{I \ni j} x_I \;=\; w_j \quad \text{for } j \in V \\ &\qquad x_I \geq 0 \qquad\qquad \text{for each } I \end{aligned} \tag{1}$$

If $\boldsymbol{w} = \mathbf{1}$ (the vector of all 1's) and if the variables x_I are restricted to integral values, then the optimal solution of the resulting integer program is just the ordinary chromatic number $\chi(G)$: Every independent set I with $x_I = 1$ forms a color class, the n equations in (1) enforce that every vertex is contained in some color class, and the objective is to use the minimum number of color classes.

The *fractional chromatic number* $\chi_f(G)$ equals $\chi_f(G, \mathbf{1})$. The chromatic number and the fractional chromatic number are closely related; see [1]. For instance, the chromatic number $\chi(G)$ is at most $\log |V|$ times $\chi_f(G)$.

The connection between the buffer scheduling problem and the fractional chromatic number is as follows: Suppose that G is the conflict graph, and that the current load vector is $\boldsymbol{w}$. Then the minimum time needed to empty all the buffers (without any new tasks arriving) equals $\chi_f(G, \boldsymbol{w})$. The variables x_I indicate the amount of time for which I should be run in this schedule.

Using this connection and amplifying the hardness of coloring over time, we show that the minimum buffer size cannot be efficiently approximated. Thus we should not hope that simple greedy-type (on-line or off-line) algorithms will have good worst case ratios on *all* graphs.

Theorem 2.1. *For any $c > 0$, the buffer minimization problem cannot be approximated within an $O(n^c)$ factor in polynomial time, unless $P = NP$.*

Proof. We use the following theorem of Lund and Yannakakis [6]: There is a $c > 0$ for which the fractional chromatic number $\chi_f(G)$ of an n-vertex graph G cannot be approximated within an $O(n^c)$ factor in polynomial time, unless $P = NP$. (Stronger results are known for coloring, but using them would not improve the theorem.)

Suppose that there exists a polynomial-time approximation algorithm $\mathcal{A}$ for buffer minimization with worst case guarantee an^c. We can use $\mathcal{A}$ to design a polynomial-time decision procedure $\mathcal{D}$ for the following problem: Given a graph $G = (V, E)$ and a positive integer F, decide whether $F < \chi_f(G)$ or $\chi_f(G) \le 2F$ (when $F < \chi_f(G) \le 2F$ then both answers are correct outputs). By repeatedly calling $\mathcal{D}$ for G and the values $F_j = 2^j$, we can sandwich $\chi_f(G)$ in polynomial time between two consecutive powers of two. This would yield a polynomial time 2-approximation algorithm for computing the fractional chromatic number, and then the result of Lund and Yannakakis [6] would imply $P = NP$.

How do we design this procedure $\mathcal{D}$? We construct a special instance of the buffer minimization problem on G. At every time jF, for $j = 0, 1, 2, \ldots, an^{c+1}$, a task $\mathbf{1}$ arrives. The size of the task sequence $\bar{t}, \bar{\boldsymbol{\tau}}$ is polynomially bounded in the size of G. We feed this instance $\langle G, \bar{t}, \bar{\boldsymbol{\tau}} \rangle$ into $\mathcal{A}$. Denote by $B^{\mathcal{A}}$ the buffer size computed by $\mathcal{A}$. If $B^{\mathcal{A}} \le an^c$, return "$\chi_f(G) \le 2F$", else return "$\chi_f(G) > F$".

To justify the correctness of $\mathcal{D}$, suppose first $B^{\mathcal{A}} \le an^c$. The total work assigned to each node is an^{c+1}, and right after the last task arrives at time Fan^{c+1} the workload in all buffers of $\mathcal{A}$ is at most an^c. Thus in time Fan^{c+1} each node processed work at least $an^{c+1} - an^c$. Therefore $\chi_f(G) \le Fan^{c+1}/(an^{c+1} - an^c) \le 2F$ (without loss of generality, we assume $n \ge 2$).

On the other hand, if $B^{\mathcal{A}} > an^c$ then the optimal buffer size is greater than 1. Thus it is not possible to process the workload $\mathbf{1}$ in time F, and thus $\chi_f(G) > F$.

3 The Online Problem for Arbitrary Conflict Graphs

Let $G = (V, E)$ be an arbitrary but fixed conflict graph. In this section we show that $\mathtt{buf}(G) < \infty$. The main idea is the following. Since an on-line algorithm does not know the current state of the off-line algorithm, it tries to choose states that would be good for *every* possible state of the off-line algorithm.

For two states $\boldsymbol{z}$ and $\boldsymbol{z}'$, define $\pi(\boldsymbol{z}, \boldsymbol{z}') = \chi_f(G, [\boldsymbol{z} - \boldsymbol{z}']^+)$, that is, $\pi(\boldsymbol{z}, \boldsymbol{z}')$ is the minimum processing time needed to reach a state $\boldsymbol{z}'' \le \boldsymbol{z}'$ from $\boldsymbol{z}$. A state $\boldsymbol{y}$ is called *off-line feasible* at time t, if there exists an off-line 1-schedule for all the requests seen till time t which is in state $\boldsymbol{y}$ at time t. A state $\boldsymbol{z}$ is called α-*universal* at time t if, for any off-line feasible state $\boldsymbol{y}$ at time t, $\pi(\boldsymbol{z}, \boldsymbol{y}) \le \alpha$. In other words, from an α-universal state we can reach any possible off-line feasible state within α time units. An algorithm is α-*universal* if at each point in time its state is α-universal. From the definition of α-universality we immediately get the following lemma.

Lemma 3.1. *Any α-universal on-line algorithm is weakly $(\alpha + 1)$-competitive.*

A linear programming lemma. Let I range over the independent sets of G. We formulate a linear program with the following intended meaning of the variables: The vector $\boldsymbol{a}$ is a state of the on-line algorithm. In the proof we think of $\boldsymbol{a}$ as the vertex weights for which we seek an optimal fractional coloring. For any feasible solution, the variables x_I define a fractional coloring of $(G, \boldsymbol{a})$. The variables u_I describe a change of this coloring that gives a fractional coloring of $(G, \boldsymbol{a} - \boldsymbol{\varepsilon})$ in which the number of colors decreases by δ. Finally, d is a crucial parameter which says that if a color was used heavily in the original coloring, it is also used in the modified coloring. All variables except the u_I are non-negative.

Let W be the set of all tuples $(\boldsymbol{a}, \boldsymbol{x}, \boldsymbol{\varepsilon}, \delta, d, \boldsymbol{u})$ satisfying the following linear constraints:

$$
\begin{aligned}
0 \le \sum_{I \ni j} x_I &= a_j \quad \text{for each } j \in V \\
0 \le \sum_{I \ni j} u_I &= \varepsilon_j \le 1 \quad \text{for each } j \in V \\
0 \le \sum_{I} u_I &= \delta \le \chi_f(G, \mathbf{1}) \\
[u]_I^+ \le x_I \text{ and } [u]_I^+ &\le d \quad \text{for each } I
\end{aligned}
$$

The key step in the proof is to show that it is never necessary to use large values of d. This implies that if we have a coloring for the current weights and change the weights a little, we can also bound the change in the coloring. In the next lemma we give show that such a bound exists.

Lemma 3.2. *Define* $f(\boldsymbol{a}, \boldsymbol{x}, \boldsymbol{\varepsilon}, \delta) = \min\{d \mid (\exists \boldsymbol{u})(\boldsymbol{a}, \boldsymbol{x}, \boldsymbol{\varepsilon}, \delta, d, \boldsymbol{u}) \in W\}$. *There exists a constant D such that the function f is upper bounded by D on all points where it is defined (i.e., finite).*

Proof. The set of tuples $(\boldsymbol{a}, \boldsymbol{x}, \boldsymbol{\varepsilon}, \delta, d)$ such that $(\boldsymbol{a}, \boldsymbol{x}, \boldsymbol{\varepsilon}, \delta, d, \boldsymbol{u}) \in W$ for some $\boldsymbol{u}$ is a polytope, since this is simply a projection of W. Thus function f, being a value of a linear program, is piecewise linear on its domain. Moreover, its domain consists of a finite number of regions in which f is linear. Consequently, it is sufficient to verify that f is bounded on any infinite feasible ray (halfline) in variables $(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{\varepsilon}, \delta)$.

Since feasible values of $\boldsymbol{\varepsilon}$ and δ are bounded, they are also bounded along any feasible ray. Since $\boldsymbol{a}$ and $\boldsymbol{x}$ are non-negative, they must be non-decreasing along any feasible ray.

Now take any two points $(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{\varepsilon}, \delta)$ and $(\boldsymbol{w}', \boldsymbol{x}', \boldsymbol{\varepsilon}', \delta')$ on the same feasible ray, so that the second one is farther away from the origin of the ray. This means that $\boldsymbol{\varepsilon}' = \boldsymbol{\varepsilon}$, $\delta' = \delta$, $\boldsymbol{a}' \ge \boldsymbol{a}$, and $\boldsymbol{x}' \ge \boldsymbol{x}$. Let $(\boldsymbol{a}, \boldsymbol{x}, \boldsymbol{\varepsilon}, \delta, d, \boldsymbol{u}) \in W$ and $(\boldsymbol{a}', \boldsymbol{x}', \boldsymbol{\varepsilon}, \delta, d', \boldsymbol{u}') \in W$ be the corresponding feasible vectors with the minimal values of d and d'. We claim that $(\boldsymbol{a}', \boldsymbol{x}', \boldsymbol{\varepsilon}, \delta, d, \boldsymbol{u}) \in W$ as well. We have $x'_I - u_I \ge x_I - u_I \ge 0$ by the feasibility of the first vector. All the other constraints follow directly by the feasibility of one of the two vectors. By minimality of d', we

have $d' \le d$. Consequently, along any infinite ray, the minimal feasible value of d cannot increase.

The algorithm. Now we define an on-line algorithm as follows. Let $\alpha = qD$, where q is the number of independent sets and D is the constant from Lemma 3.2. Suppose that $\boldsymbol{a}$ is the current state of the algorithm, and let $\boldsymbol{x}$ be an optimal fractional coloring of $\boldsymbol{a}$. Process an arbitrary independent set I with $x_I > D$ for time $x_I - D$. If no such set remains and there exists a vertex with non-zero load, we process any set containing such a vertex.

Theorem 3.3. *For any graph $G = (V, E)$, $\mathtt{buf}(G)$ is finite.*

Proof. We claim that the above algorithm is α-universal. By Lemma 3.1 it then follows that $\mathtt{buf}^-(G) \le \alpha + 1$, and using doubling we obtain $\mathtt{buf}(G) \le 4\alpha + 4$.

By the definition of the algorithm, if the load vector is non-zero, some non-trivial set is processed. Thus on any finite request sequence the zero load vector is eventually achieved and a feasible schedule is generated.

It remains to verify that the algorithm is in an α-universal state at any time. It is clearly true at time 0. When a task arrives, off-line and on-line loads change by the same amount. So if the algorithm was in an α-universal state right before a task arrival, it will also be in an α-universal state afterwards. Thus we only need to verify that the schedule remains in an α-universal state when processing an independent set during an interval when no task arrives.

If there is no set with weight larger than D in $\boldsymbol{x}$, we can reach $\mathbf{0}$ in time at most $qD = \alpha$, thus the current state and any following state (before arrival of the next task) is trivially α-universal.

The remaining case is when an independent set I with weight $w_I \ge D + \beta$ is processed for some time $\beta > 0$. Suppose that during this time the algorithm changes its state from $\boldsymbol{a}$ to $\boldsymbol{a}'$, while the adversary changes its state from $\boldsymbol{y}$ to $\boldsymbol{y}'$. We need to verify that $\pi(\boldsymbol{a}, \boldsymbol{y}) \le \alpha$ implies $\pi(\boldsymbol{a}', \boldsymbol{y}') \le \alpha$.

Trivially, $\pi(\boldsymbol{a}, \boldsymbol{y}') \le \pi(\boldsymbol{a}, \boldsymbol{y}) + \pi(\boldsymbol{y}, \boldsymbol{y}') \le \alpha + \beta$. To conclude the proof it is sufficient to show that there exists an optimal fractional coloring of $[\boldsymbol{a} - \boldsymbol{y}']^+$ with weight of I at least β. For if we have such a coloring and decrease the weight of I by β, we obtain a coloring of $[\boldsymbol{a}' - \boldsymbol{y}']^+$. Since $\boldsymbol{a}'$ is obtained from $\boldsymbol{a}$ by processing I for time β, this coloring of $[\boldsymbol{a}' - \boldsymbol{y}']^+$ has weight at most α.

Let $\boldsymbol{\varepsilon} = \boldsymbol{a} - [\boldsymbol{a} - \boldsymbol{y}']^+$, that is $\varepsilon_i = \min\{a_i, y'_i\}$ for each i. Choose $\boldsymbol{u}$ such that $\boldsymbol{x} - \boldsymbol{u}$ is an optimal fractional coloring of $\boldsymbol{a} - \boldsymbol{\varepsilon}$, and define $\delta = \chi_f(G, \boldsymbol{a}) - \chi_f(G, \boldsymbol{a} - \boldsymbol{\varepsilon})$. Clearly, $\mathbf{0} \le \boldsymbol{\varepsilon} \le \mathbf{1}$ and $\delta \le \chi_f(G, \boldsymbol{\varepsilon}) \le \chi_f(G, \mathbf{1})$. By inspection of the linear program, if d is sufficiently large then $(\boldsymbol{a}, \boldsymbol{x}, \boldsymbol{\varepsilon}, \boldsymbol{u}, \delta, d) \in W$. Lemma 3.2 implies that there are $\boldsymbol{u}'$ and $d' \le D$ for which $(\boldsymbol{a}, \boldsymbol{x}, \boldsymbol{\varepsilon}, \delta, d', \boldsymbol{u}') \in W$. Then $\boldsymbol{x} - \boldsymbol{u}'$ is also an optimal fractional coloring of $\boldsymbol{a} - \boldsymbol{\varepsilon}$ and $x_I - u'_I \ge x_I - d' \ge x_I - D \ge \beta$.

4 The Online Problem on Complete k-Partite Graphs

At each step, a scheduling algorithm needs to determine an independent set I of processors that should execute their tasks. Algorithm Greedy determines I in the most obvious way: it iteratively chooses the processor with highest load,

and eliminates its neighbors. To define GREEDY formally, we need to be a bit careful, as the time is continuous and ties need to be appropriately resolved. Denote GREEDY's load vectors by $\boldsymbol{a}$. We view the computation as being divided into ϵ-steps, with $\epsilon \to 0$. At each such ϵ-step, GREEDY determines I as follows: Start with $I = \emptyset$. Iteratively pick v with maximum a_v, add v to I, and remove v and its neighbors from G. Stop when $G = \emptyset$. Then, for all $v \in I$, decrease a_v by ϵ. Consider a time interval $[t, t']$ in which no tasks are issued. Divide it into intervals of length ϵ. Determine the state of GREEDY at time t', and take its limit for $\epsilon \to 0$. For complete k-partite graphs this limit is always well-defined. For such graphs, if there are j color classes that contain a node with maximum buffer size, then GREEDY will process the buffers of all nodes in these color classes at speed $1/j$.

Throughout this section, by a *subgraph* X we mean the subgraph of G induced by X. By $\boldsymbol{z}$ we denote an off-line state, and by B the optimal buffer size. Since GREEDY does not depend on B, we can assume that $B = 1$.

Smooth subgraphs. If X is a vertex set, then denote $N(X) = \bigcup_{v \in X} N(v) - X$. We say that X is *smooth* if all vertices in X have the same neighbors outside X that is, $N(v) - X = N(X)$ for $v \in X$.

GREEDY's behavior on smooth subgraphs is easy to characterize. Suppose X is a smooth subgraph with neighborhood $L = N(X)$, and assume that there are no tasks issued between the current time t and some time $t' > t$. If $\max_X \boldsymbol{a} > \max_L \boldsymbol{a}$ then, when GREEDY chooses its independent set I, it will always pick at least one vertex from X, namely one that realizes the maximum $\max_X \boldsymbol{a}$, and it will not include any vertices from L. So $\max_X \boldsymbol{a}$ will keep decreasing while $\max_L \boldsymbol{a}$ will stay the same. On the other hand, if $\max_X \boldsymbol{a} \le \max_L \boldsymbol{a}$, then this inequality will remain true until time t', since, by the previous statement, for any choice of ϵ-steps, $\max_X \boldsymbol{a}$ cannot exceed $\max_L \boldsymbol{a}$ by more than ϵ.

Lemma 4.1. *Suppose $K \subseteq V$ is a smooth clique in G with $N(K) = L$. Then*

$$\Sigma_K \boldsymbol{a} \le \Sigma_K \boldsymbol{z} + |K| \max_L \boldsymbol{a}. \tag{2}$$

Proof. That (2) is preserved when tasks are issued is obvious. So consider task execution. If $\max_K \boldsymbol{a} \le \max_L \boldsymbol{a}$ then $\Sigma_K \boldsymbol{a} \le |K| \max_K \boldsymbol{a} \le |K| \max_L \boldsymbol{a}$, so (2) holds. Further, if $\max_K \boldsymbol{a} = \max_L \boldsymbol{a}$, then this equality will be preserved throughout until the next task arrives. If $\max_K \boldsymbol{a} > \max_L \boldsymbol{a}$ then, at the next infinitesimal ϵ-step, the independent set I used by GREEDY contains exactly one node from K and is disjoint with L. Then the left-hand side of (2) decreases by ϵ and the right-hand side cannot decrease by more than ϵ.

Lemma 4.2. *Suppose that $X \subseteq V$ is a smooth, complete k-partite subgraph of G with color classes $J_1, \ldots, J_k$ and $N(X) = L$. Then*

$$\sum_{i=1}^{k} \max_{J_i} [\boldsymbol{a} - \boldsymbol{z}]^+ \le \max \{ k \max_L \boldsymbol{a} , k - 1 \} \tag{3}$$

Proof. Inequality (3) is preserved when tasks are issued, so it is sufficient to consider task execution. If $\max_X \boldsymbol{a} \le \max_L \boldsymbol{a}$, then the left-hand side is at most $\sum_{i=1}^{k} \max_{J_i} \boldsymbol{a} \le k \max_X \boldsymbol{a} \le k \max_L \boldsymbol{a}$. Further, if $\max_X \boldsymbol{a} = \max_L \boldsymbol{a}$ then this equality will remain true throughout task execution. Suppose now $\max_X \boldsymbol{a} > \max_L \boldsymbol{a}$. In the next ϵ-step GREEDY will use an independent set I such that $I \cap (X \cup L) = J_j$ for some J_j that maximizes $\max_{J_j} \boldsymbol{a}$. We have two sub-cases. If $a_v \le z_v$ for all $v \in J_j$ then $a_v \le 1$ for all $v \in X$. So the jth term on the left-hand side is 0 and the other terms are at most 1. Overall, the left-hand side is at most $k-1$. Finally, suppose that $a_v > z_v$ for some $v \in J_j$. All positive $a_v \in J_j$ will decrease by ϵ, decreasing the left-hand side by ϵ. There can only be one j' for which some $z_v \in J_{j'}$ decreases, increasing the left-hand side by at most ϵ. So the left-hand side cannot increase and the right-hand side does not change.

Theorem 4.3. *For the complete graph* K_n, $\mathtt{buf}(K_n) = \mathtt{buf}^-(K_n) = H_n$.

Proof. (Lower bound) The adversary strategy consists of phases. Before phase p starts, the following invariant holds: there is a set X of $n-p+1$ processors such that $z_i = 0$ for $i \in X$ and $\Sigma_X \boldsymbol{a} \ge (n-p+1)(H_n - H_{n-p+1})$. The adversary creates task 1 for processors $i \in X$ and waits for time $n-p$. The new buffers satisfy $\Sigma_X \boldsymbol{a}' \ge \Sigma_X \boldsymbol{a} + (n-p+1) - (n-p) = \Sigma_X \boldsymbol{a} + 1$. Pick j for which a'_j is minimum and let $X' = X - \{j\}$. Then $\Sigma_{X'} \boldsymbol{a}' \ge \frac{n-p}{n-p+1}(\Sigma_X \boldsymbol{a} + 1) \ge (n-p)(H_n - H_{n-p})$. The adversary can zero all z_i for $i \in I'$. Thus the invariant is preserved.

In phase n, the only processor in X will have workload at least $a_i = H_n - 1$ in the buffer, so after adding 1 to processor i, the workload will reach H_n.
(Upper bound) We prove that GREEDY is H_n-competitive. Order the processors so that $a_1 \ge a_2 \ge \ldots \ge a_n$. By (2), for each j we have $\sum_{i \le j} a_i \le j + j a_{j+1}$, where for $j = n$ we assume that $a_{n+1} = 0$. Multiply the jth inequality, for $j < n$, by $1/j(j+1)$, multiply the nth inequality by $1/n$, and then add all the inequalities together. We get $a_1 \le H_n$, and the upper bound follows.

Theorem 4.4. *If* G *is a complete* k*-partite graph then* $\mathtt{buf}(G) \le H_{k-1} + 1$.

Proof. We prove that GREEDY is $(H_{k-1}+1)$-competitive. Let the color classes of G be $J_1, J_2, \ldots, J_k$. Let $A_i = \max_{J_i} \boldsymbol{a}$, for all i. Reorder the color classes so that $A_1 \ge A_2 \ge \ldots \ge A_k$. Then, (3) implies that $\sum_{i=1}^{j} A_i \le \max\{jA_{j+1}, j-1\} + j$ for each $j = 1, \ldots, k$. Pick the smallest $l \le k$ for which $lA_{l+1} \le l-1$. For $j = 1, \ldots, l-1$, multiply the jth inequality by $1/j(j+1)$, multiply the lth inequality by $1/l$, and then add the first l inequalities together. We get

$$\sum_{j=1}^{l-1} \frac{1}{j} \sum_{i=1}^{j} A_i - \sum_{j=1}^{l-1} \frac{1}{j+1} \sum_{i=1}^{j} A_i + \frac{1}{l} \sum_{i=1}^{l} A_i \le \sum_{j=1}^{l-1} \frac{1}{j+1} A_{j+1} + H_{l-1} + 1,$$

which yields $A_1 \le H_{l-1} + 1 \le H_{k-1} + 1$, and the theorem follows.

Our analysis of GREEDY in Theorem 4.4 is tight. For the lower bound, we use a complete graph with one edge missing, say $G = K_{k+1} - \{(1,n)\}$. Suppose that we have a configuration $\boldsymbol{a} = (\alpha, \beta, \beta, \ldots, \beta)$ and $\boldsymbol{z} = \boldsymbol{0}$, where $\alpha \le \beta \le 1 - 1/k$.

Initially, $\alpha = \beta = 0$. Create task $(0, 1, 1, \ldots, 1)$, process for time $k-1$, then create task $(1, 0, 1, \ldots, 1)$, and process for time k. The adversary can zero all his buffers. At the end GREEDY will be in configuration $(\beta', \alpha', \beta', \ldots, \beta')$, where $\alpha' = \beta - (1 - 1/k)(1 - 1/k - \beta)$, $\beta' = \beta + (1 - 1/k - \beta)/k$. Note that $\alpha' \leq \beta' \leq 1 - 1/k$. We can now repeat the process with the nodes 1 and n switched. Thus in the limit, β will converge to $1 - 1/k$. Then we can use the strategy for K_k to increase the buffers size to $H_k + 1 - 1/k = H_{k-1} + 1$.

The next theorem shows that the upper bound achieved by GREEDY on complete k-partite graphs is tight within a small additive factor. Note that as a special case of this lower bound, we obtain that $\mathtt{buf}^-(P_3) \geq 2$. The proof involves a somewhat tedious adversary argument, and is omitted in this abstract.

Theorem 4.5. *Consider a complete k-partite graph G in which μ of the independent sets in the k-partition consist of a single vertex, whereas the remaining $k - \mu$ independent sets all have at least two vertices. If $\mu = 0$, then $\mathtt{buf}^-(G) \geq H_{k-1} + 1$. If $\mu \geq 1$, then $\mathtt{buf}^-(G) \geq H_{k-1} + (k - \mu)/(k - 1)$.*

5 The Online Problem on Trees

In this section we prove that the strong competitive ratio for trees of diameter Δ is at most $1 + \Delta/2$. In particular, $\mathtt{buf}(P_n) \leq (n + 1)/2$.

Let $G = (V, E)$ be a graph and $\mathcal{A}$ an on-line algorithm for an induced subgraph H of G. The *greedy extension* of $\mathcal{A}$, denoted $\mathrm{GE}(\mathcal{A})$ is an algorithm for G that works like this: if $\mathcal{A}$ processes a set I at a given step, then $\mathrm{GE}(\mathcal{A})$ chooses greedily (that is, choosing nodes with largest buffers) a maximal independent set $J \subseteq V - I - N(I)$, and processes $I \cup J$.

For certain graphs G and its subgraphs H, we can estimate the relationship between the competitive ratios of $\mathcal{A}$ and $\mathrm{GE}(\mathcal{A})$.

Lemma 5.1. *Suppose that G is constructed from H by adding a number of new vertices of degree 1, each connected by a new edge to some vertex in H. Then $\mathtt{buf}(G) \leq \mathtt{buf}(H) + 1$ and $\mathtt{buf}^-(G) \leq \mathtt{buf}^-(H) + 1$.*

Proof. Let $\mathcal{A}$ be c-competitive on H (the proof for weak competitiveness is the same). Let $\mathcal{B}$ be $\mathrm{GE}(\mathcal{A})$. We can assume the buffer size is $B = 1$, since $\mathcal{B}$ does not use the information about the off-line buffer size, unless $\mathcal{A}$ does so.

By the definition of $\mathcal{B}$, its behavior on H is exactly the same as that of $\mathcal{A}$. In particular, the buffer size of any vertex of H is at most c at any time.

Consider $v \in G - H$, and let w be its unique neighbor in H. Let $\boldsymbol{a}$ and $\boldsymbol{z}$ be the states of $\mathcal{B}$ and the optimal off-line algorithm, respectively. We claim that the following invariant holds at all times:

$$a_v + a_w \leq z_v + z_w + c - 1. \tag{4}$$

This inequality must be true whenever $a_v = 0$, since otherwise, by putting load $1 - z_w$ at w we would contradict the c-competitiveness of $\mathcal{A}$ on G. If a new task arrives, both sides of (4) increase by the same amount, thus the inequality is

preserved. If some independent set is processed for time ϵ while $a_v > 0$, then the left-hand side of (4) decreases by ϵ, by the definition of the algorithm B, and the right-hand side cannot decrease by more than ϵ. Thus (4) is preserved. Since $z_v, z_w \leq 1$, we obtain $a_v \leq a_v + a_w \leq z_v + z_w + c - 1 \leq c + 1$.

Theorem 5.2. *For any tree T with diameter Δ, $\mathtt{buf}(T) \leq 1 + \Delta/2$. In particular, $\mathtt{buf}(P_n) \leq (n+1)/2$.*

Proof. By iteratively adding leaves, in Δ steps we can obtain any tree with diameter 2Δ from K_1, and any tree with diameter $2\Delta + 1$ from K_2. The bound follows by iterating the lemma and noting that $\mathtt{buf}(K_1) = 1$ and $\mathtt{buf}(K_2) = 1.5$.

Note that Theorem 5.2 gives another 2-competitive algorithm for P_3: If there is any load on the middle vertex, run this vertex, otherwise run the two endpoints.

6 The Online Problem on Small Graphs

In this section we discuss competitive ratios of the ten connected graphs with up to four vertices. By Theorem 4.3, the complete graphs K_1, K_2, K_3, and K_4 have competitive ratios 1, $\frac{3}{2}$, $\frac{11}{6}$, and $\frac{25}{12}$, respectively. By Theorems 4.4 and 4.5, the complete bipartite graphs $K_{1,2}$ $(= P_3)$, $K_{1,3}$, and $K_{2,2}$ $(= C_4)$ have competitive ratio 2. All these bounds are attained by algorithm Greedy. The corresponding weak competitive ratios are the same. Thus for all these graphs the problem is completely solved.

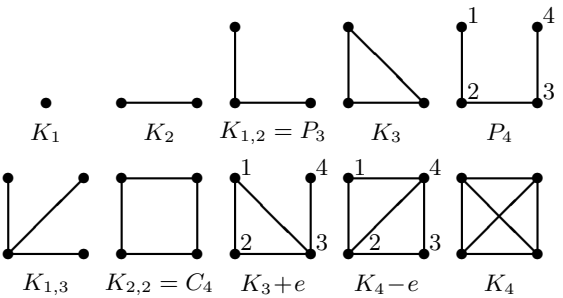


Fig. 1. The connected graphs with at most four vertices.

The three remaining graphs are the path P_4, the triangle plus an edge $K_3{+}e$, and "diamond graph" $K_4{-}e$, see Figure 1. Since all these graphs contain P_3, their weak competitive ratio is at least 2. For these graphs, we can prove the following bounds (the proofs will appear in the full version of the paper):

$$\begin{aligned} 2 &\leq \mathtt{buf}^-(P_4) \leq \mathtt{buf}(P_4) \leq \tfrac{5}{2} \\ \tfrac{13}{6} &\leq \mathtt{buf}^-(K_3{+}e) \leq \mathtt{buf}(K_3{+}e) \leq \tfrac{5}{2} \\ 2 &= \mathtt{buf}^-(K_4{-}e) \leq \mathtt{buf}(K_4{-}e) \leq \tfrac{5}{2} \end{aligned}$$

7 Final Comments

The main open problem is to establish tighter bounds on the competitive ratios for general graphs. The first step may be to either give a polynomial upper bound or a super-logarithmic lower bound, if any of these is possible.

For trees, we were unable to prove any lower bound better than 2. We suspect that there may be an algorithm for paths, and possibly for trees as well, with a constant competitive ratio (independent of n).

All algorithms we presented in the paper are *memoryless*, that is, they don't keep track of the past history. The behavior of such an algorithm depends only on its current buffer loads. We believe that algorithms that use information about possible adversary configurations can achieve better competitive ratios. Using the history, for any B, we can compute all possible adversary configurations that can be reached with buffer size up to B. However, the question of how to represent and use this information appears itself to be a difficult problem (in fact, we proved that maintaining this information is NP-hard). Perhaps, instead of keeping track of the whole history, it is sufficient to maintain only some lower bounds on the buffer sizes. It is quite easy to define lower bounds using complete subgraphs, for example.

A natural starting point for the above investigations would be to analyze the competitive ratios for small graphs, and for P_4 in particular. We made some progress in this direction, but many questions remain open. A complete analysis of small graphs would give good insight into the problem and may provide new ideas for the general case.

Acknowledgements. This research was partially supported by: NSF grant CR-9988360, grant A1019901 of GA AV ČR, postdoctoral grant 201/97/P038 of GA ČR, project LN00A056 of MŠMT ČR, START program Y43-MAT of the Austrian Ministry of Science, and a fellowship of the Pál Erdős Institute Budapest.

References

1. M. Grötschel, L. Lovász, and A. Schrijver [1981]. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica* 1, 169–197.
2. S. Irani and V. Leung [1997]. Probabilistic analysis for scheduling with conflicts, *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, 286–295.
3. S. Irani and V. Leung [1996]. Scheduling with conflicts, and applications to traffic signal control, *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, 85–94.
4. A. Bar-Noy, M. Bellare, M.M. Halldórsson, H. Shachnai and T. Tamir [1998]. On chromatic sums and distributed resource allocation. *Information and Computation* 140, 183–202.
5. A. Bar-Noy and G. Kortsarz [1998]. Minimum color sum of bipartite graphs, *Journal of Algorithms* 28, 339-365.

6. C. Lund and M. Yannakakis [1994]. On the hardness of approximating minimization problems. *Journal of the ACM* 41, 960–981.
7. R. Motwani, S. Philips and E. Torng [1994]. Non-clairvoyant scheduling, *Theoretical Computer Science* 130, 17–47.
8. E. Kubicka and A.J. Schwenk [1989]. An introduction to chromatic sums, *Proc. ACM Computer Science Conference*, 39–45.
9. N.A. Lynch [1996]. Distributed Algorithms. Morgan Kauffman Publishers, San Francisco, California, 1996.

On Minimizing Average Weighted Completion Time of Multiprocessor Tasks with Release Dates*

Aleksei V. Fishkin, Klaus Jansen[1], and Lorant Porkolab[2]

[1] Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel, Olshausenstrasse 40, 24 098 Kiel, Germany, {avf,kj}@informatik.uni-kiel.de

[2] Applied Decision Analysis, PricewaterhouseCoopers, 1 Embankment Place, London WC2N 6RH, United Kingdom, lorant.porkolab@uk.pwcglobal.com

Abstract. We consider the problem of scheduling n independent multiprocessor tasks with release dates on a fixed number of processors, where the objective is to compute a non-preemptive schedule minimizing the average weighted completion time. For each task, in addition to its processing time and release date, there is given a prespecified, dedicated subset of processors which are required to process the task simultaneously. We propose here a polynomial-time approximation scheme for the problem, making substantial improvement on previous results and following the recent developments [1,2,15] on approximation schemes for scheduling problems with the average weighted completion time objective.

1 Introduction

In the traditional theory of scheduling, each task is processed by only one processor at a time. However, new theoretical approaches have emerged to model scheduling on parallel architectures. One of these is scheduling multiprocessor tasks, see e.g. [9,12].

In this paper we consider the problem of scheduling dedicated tasks with release dates on a set of processors in order to minimize the average weighted completion time, denoted as $P|fix_j, r_j|\sum w_jC_j$. Formally, we are given a set of n tasks $T = \{0, 1, \ldots, n-1\}$ and a set of m processors $M = \{1, 2, \ldots, m\}$. Each task $j \in T$ has a *processing time* p_j, a *release date* r_j, a positive *weight* w_j and a prespecified processor subset $\tau_j \subseteq M$, called the *type* of j. Processing of each task can start only after its release date and once started must be completed without interruption. Each processor can work on at most one task at a time, each task must be processed simultaneously by all processors of τ_j. The objective is to minimize *the average weighted completion time* $\sum w_jC_j$, where C_j denotes the completion time of task j.

* Supported in part by DFG - Graduiertenkolleg "Effiziente Algorithmen und Mehrskalenmethoden" and by EU project APPOL "Approximation and On-line Algorithms", IST-1999-14084

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 875–886, 2001.

Previous results: The first polynomial-time approximation scheme (PTAS) for a strongly NP-hard scheduling problem minimizing the average weighted completion time was given for scheduling jobs with no release dates on identical parallel machines $P||\sum w_jC_j$ [15]. Then recently it was proved in [1] that there are polynomial-time approximation schemes for many different variants of classical scheduling problems with release dates and average weighted completion time objective. These include scheduling on identical parallel machines $P|r_j|\sum w_jC_j$ or on a fixed number of unrelated machines $Rm|r_j|\sum w_jC_j$ with and without preemptions, but exclude models for multiprocessor tasks.

In the multiprocessor setting, variants of the problem have been studied, but the previous research has mainly focused on the objectives of minimizing *the makespan* $C_{\max} = \max_{j=0}^{n-1} C_j$ and *the sum of completion times* $\sum C_j$. Regarding the worst-case time complexity, it is known that $P3|fix_j|C_{\max}$ [14] is strongly NP-hard. Problem $P|fix_j|\sum C_j$ was first studied in [14] and shown to be strongly NP-hard even when all tasks have unit execution times. If the number of processors m is fixed, then the problem $Pm|fix_j, p_j = 1|\sum C_j$ (with unit execution times) becomes polynomial-time solvable [6] even if the tasks have release dates. The negative result was strengthened in [7], where the authors proved that already problem $P2|fix_j|\sum C_j$ is strongly NP-hard. Contrasting this with the fact that for identical parallel machines only the general problem $P||\sum C_j$ is strongly NP-hard, while $Pm||\sum C_j$ is just weakly NP-hard, indicates that computing optimal and/or approximate schedules for dedicated tasks is likely to be much harder than the corresponding (classical) non-multiprocessor variants. However, there are PTASs for both $Pm|fix_j|C_{\max}$ [3] and $Pm|fix_j|\sum C_j$ [2].

The above results are, in some sense, the strongest possible ones someone can expect. First, it shows the existence of a PTAS for a problem with fixed parameter m that cannot have a fully PTAS [11]. Second, following the ideas in [14] and by using the results [4,5,10] one can prove that both $P|fix_j, p_j = 1|C_{max}$ and $P|fix_j, p_j = 1|\sum C_j$ cannot be approximated within a factor of $m^{\frac{1}{2}-\varepsilon}$, neither for some $\varepsilon > 0$, unless P=NP; nor for any $\varepsilon > 0$, unless NP=ZPP. Hence, not only the above results cannot be extended or generalized for the general variant where m is not fixed, but even substantially weaker approximation results cannot be expected.

New results: By combining various ideas from [1,2,3,8,13], we provide here a generalization of several results and obtain a PTAS for $Pm|fix_j, r_j|\sum w_jC_j$. In order to be able to cope with multiprocessor tasks, we refine some recent sophisticated approximation techniques for the average completion time objective. These include the interval time-partitioning technique developed in [8,13] and, the techniques of geometric rounding and time stretching presented in [1]. We also employ a well known idea of *transformations* – simplify instances and schedules with some loss in the objective. We make adjustments of release dates and processing times classifying the tasks as huge and tiny. In our approach, we also apply some recent makespan minimization tools from [3] to schedule tasks within single intervals. For handling tiny tasks, we use a linear programming formulation along with some rounding, and observe that in some near optimal

schedule tiny tasks of the same type can be scheduled by Smith's rule [16]. Then, by using the task delaying technique presented in [1], we introduce special *compact instances* in which there is only a constant number of tasks at any release date. In order to use use dynamic programming which integrates all the previous components, we introduce block-superblock structure and show that any task can be scheduled within only a constant number of intervals after release. This requires the creation of a sequence of gaps in the schedule, where all processors are idle for a certain period of time. The obtained PTAS is a combination of the instance transformations and the dynamic programming algorithm.

By an appropriate combination of these ideas and a careful analysis of the algorithms, we prove the following result.

Theorem 1. *There is PTAS for* $Pm|fix_j, r_j|\sum w_jC_j$ *that computes for any fixed* m *and* $\epsilon > 0$ *accuracy,* $(1+\varepsilon)$*-approximate solutions in* $O(n \log n)$ *time.*

The paper is organized as follows: In Section 2, we introduce some notations, give definitions, and describe several transformations (that can provide simplifications of instances, schedules and the objective function). In addition, we show some useful properties of these transformations. In Section 3, we adopt the PTAS for $Pm|fix_j|C_{\max}$ [3] and discuss some consequences. Finally, in Section 4, we present a dynamic programming framework. However, due to space limitation, here we omit many technical details leaving them to the full version of the paper.

2 Preliminaries

A schedule σ is given by a *starting time* S_j for each task $j \in T$. Then, the value $C_j := S_j + p_j$ is the completion time of task j in σ. The average weighted completion time for σ is equal to $C_w(\sigma) := \sum w_jC_j$. We write $\sigma(I)$ to denote a feasible schedule σ with respect to instance I. A schedule $\sigma_{opt}(I)$ is called *optimal* if $C_w(\sigma_{opt}(I)) = OPT(I)$, where $OPT(I)$ is the minimal average weighted completion time for I.

Given a schedule σ, a schedule σ' is called an ε*-schedule* of σ if $C_w(\sigma')/C_w(\sigma) \leq 1 + K\varepsilon$, where K is some constant. We say that one can transform *with* $1 + O(\varepsilon)$ *loss* and in polynomial time an instance I into instance I', if the following holds: I' is obtained from I in $p(|I|)$ operations for some polynomial $p(.)$, any feasible schedule of I' is also feasible of I and, $OPT(I')/OPT(I) \leq 1+K\varepsilon$, where K is a constant. Note that the superposition of two instance transformations is also an instance transformation of the above form. Our approach is as follows. We always aim to find ε-schedules satisfying some simplifying properties. For a given instance I and a fixed accuracy $\varepsilon > 0$, we perform several instance transformations such that for the final instance there exists an ε-schedule which is amenable to a fast dynamic programming procedure. Formally, we will show below that with $1+O(\varepsilon)$ loss, I can be transformed in $O(n \log n)$ time into I' such that one can find in $O(n)$ time an ϵ-schedule of an optimal schedule $\sigma_{opt}(I')$. Clearly, this suffices to obtain a PTAS for our original problem.

For any fixed accuracy $\varepsilon > 0$, we assume w.l.o.g. that $\log_{1+\varepsilon}(1+\frac{1}{\varepsilon})$ and $\frac{1}{\varepsilon}$ are integral, and that $\varepsilon \leq 1/(m+1)^{m+2}$. We partition the time interval $(0,\infty)$ into disjoint intervals I_x of the form $[R_x, R_{x+1})$, where $R_x = (1+\varepsilon)^x$ and $x \in \mathbb{Z}$. Notation I_x will also be used to refer to the length εR_x of the interval, thus $I_x = R_{x+1} - R_x = \varepsilon R_x$ and $I_{x+1} = (1+\varepsilon)I_x$.

For an instance I and task $j \in T$, let $x(j)$ be the index for which $r_j \in I_{x(j)}$. Accordingly, $I_{x(j)}$ is called the *release* interval of j, and it is said that j is *released* in $I_{x(j)}$. Then, a task j is called *huge* if $p_j \geq \varepsilon^2 I_{x(j)}/q^*$, and *tiny* if $p_j < \varepsilon^2 I_{x(j)}/q^*$, where the parameter $q^* = q^*(m,\varepsilon) > 1$ is specified later in Section 3. For a schedule σ, let $y(j) \leq z(j)$ be those indices for which $S_j \in I_{y(j)}$ and $C_j \in I_{z(j)}$, respectively. Then, a task j is called *crossing* if $z(j) - y(j) \geq 1$, and *non-crossing* if $y(j) = z(j)$.

For simplicity, we will use the following notations throughout the paper. Let 2^M be the set of all tasks types. For a task set $\mathcal{X} \subseteq T$, $p(\mathcal{X}) := \sum_{T_j \in \mathcal{X}} p_j$ is the total processing time of the tasks in $\mathcal{X}$, and $\mathcal{X}^\tau$ is the set consisting of the tasks in $\mathcal{X}$ of type $\tau \in 2^M$. To indicate that $\mathcal{X}$ is associated with a particular instance I, we use notation $\mathcal{X}(I)$. We will denote the sets of huge and tiny tasks by $\mathcal{HT}$ and $\mathcal{TT}$, respectively. Furthermore, we will also use refinements of these notations, where $\mathcal{TT}_x$ and $\mathcal{HT}_x$ will stand for the sets of tiny and huge tasks, respectively, that are released in interval I_x.

Proposition 1. *For any instance I, one can replace, with at most $1+\epsilon$ loss, the original objective function $\sum w_j C_j$ by $\sum w_j R_{z(j)}$.*

We assume now that σ is given, and describe some modifications of it, called *techniques*, each of which is leading to a new feasible schedule σ' with some useful properties:

Stretching: Set $S'_j = (1+\varepsilon)C_j - p_j$ for each task $j \in T$. This generates εp_j idle time on τ_j.

Rearranging: Set S'_j for each task $j \in T$ such that a new schedule σ' is feasible and $z'(j) = z(j)$, i.e. tasks are rescheduled preserving $C'_j \in I_{z(j)}$. This gives us the ability to rearrange tasks inside intervals.

Shifting*: Set $S'_j = S_j - R_{y(j)} + R_{y(j)+1}$ for each task $j \in T$, i.e. in σ' we have $y'(j) = y(j)+1$, and hence the distance between S_j and the beginning of interval $I_{y(j)}$ is preserved. This generates $\varepsilon(R_{z(j)} - R_{y(j)})$ additional idle time on τ_j after the end of j.

Shifting**: Set $S'_j = S_j - R_{z(j)} + R_{z(j)+1}$ for each task j, i.e. $z'(j) = z(j)+1$, and hence the distance between C_j and the end of interval $I_{z(j)}$ is preserved. This generates $\varepsilon(R_{z(j)} - R_{y(j)})$ additional idle time on τ_j before the start of j.

Proposition 2. *If a schedule σ' is obtained from a feasible schedule σ by stretching, rearranging, or both types of shifting then σ' is an ε-schedule of σ.*

By using the above techniques we prove the following two lemmas that will be used throughout the paper.

Lemma 1. *For any feasible schedule σ there is an ϵ-schedule σ' of σ such that the following holds in σ': each task j starts not earlier than εp_j and crosses at most a constant number $s^*(\varepsilon) := \log_{1+\varepsilon}(1+\frac{1}{\varepsilon})$ of intervals; each task j with*

$p_j \leq \varepsilon^2 I_{y'(j)}$ *is non-crossing; each crossing task* j *starts at one of the points* $R_{y'(j)} + i\varepsilon^2 I_{y'(j)}$, *where* $i \in \{0, 1, \ldots, \frac{1}{\varepsilon^2} - 1\}$.

Lemma 2. *With* $1 + O(\varepsilon)$ *loss one can transform in* $O(n)$ *time an instance* I *into instance* I' *such that for each task* $j \in T$ *the following holds in* I'*: processing time* p'_j *is equal to an integer power of* $1 + \varepsilon$ *and is bounded by* $1/\varepsilon^3$ *times the length* $I_{x'(j)}$ *of release interval; release date* r'_j *is equal to the left boundary* $R_{x'(j)}$ *of release interval. Furthermore, in* I' *the quotients* p'_j/w'_j *are different for all tasks in* T.

Corollary 1. *For any instance* I, *one needs to consider at most* n/ε^7 *relevant intervals.*

Proof. By Lemma 2, $p_j \leq I_{x(j)}/\varepsilon^3$ for each task j. Suppose that all tasks are delayed by $1/\varepsilon^6$ intervals after release. Then, by Lemma 1 any task j has to be non-crossing in I_x, $x \geq x(j) + \frac{1}{\varepsilon^6}$. Thus, to schedule such tasks one needs at most n additional intervals. Hence, the schedule takes at most $n + \frac{n}{\varepsilon^6} \leq n/\varepsilon^7$ relevant intervals. □

3 Scheduling within Single Intervals

In the first part of this section, we consider the problem of scheduling in a single interval I_x. More precisely, we present an algorithm that schedules non-crossing tasks with respect to a known schedule for the crossing tasks. In order to achieve this, we generalize the PTAS for $Pm|fix_j|C_{\max}$ [3]. Since this part involves some lengthy technical details, we omit them here (due to space limitation) giving only a general description of the algorithm. However, based on some features of the algorithm, we define the value of parameter $q^* = q^*(m, \varepsilon)$ for tiny tasks, and consider the algorithm as a subroutine of the rearranging technique. After that, we show that tiny tasks are all small corresponding to intervals and can be scheduled by Smith's rule [16]. In the last part of this section, we show how an instance of our problem can be transformed into a *compact* instance in which at any release date there can only be a constant number of tasks.

3.1 Long and Short Tasks, Snapshots, and Relative Schedules

Let $\sigma(I)$ be a feasible schedule for instance I. For an interval I_x of σ, the following sets of tasks (running in I_x) are defined: set $\mathcal{W}_x$ of non-crossing tasks and set $\mathcal{K}_x$ of crossing tasks. Suppose that we have applied shifting** to σ and obtained $\sigma'(I)$. Assume w.l.o.g. that the tasks of $\mathcal{W}_x$ and $\mathcal{K}_x$ run in the interval I_{x+1}. Then, in the new schedule σ' there is at least εI_x idle time on the processors, between the tasks of $\mathcal{W}_x$ and the outgoing tasks of $\mathcal{K}_x$ or between the tasks of $\mathcal{W}_x$ and the end of interval R_{x+2}.(For illustration, see Figure 1.)

Now, the goal is to find a feasible schedule of $\mathcal{W}_x$ inside interval I_{x+1} with respect to the schedule of $\mathcal{K}_x$ in $\sigma'(I)$. This can be done as follows. First, we put the $k_x(m, \varepsilon)$ longest non-crossing tasks of $\mathcal{W}_x$ into set $\mathcal{L}_x$, and the rest into $\mathcal{S}_x$.

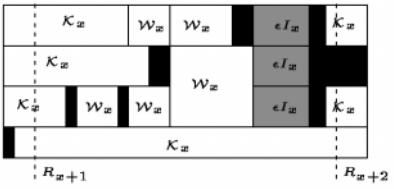


Fig. 1. Interval I_{x+1} in $\sigma'(I)$

Accordingly, we call the tasks of $\mathcal{L}_x$ *long* and the tasks of $\mathcal{S}_x$ *short*. Then, let $\mathcal{J}_x := \mathcal{L}_x \cup \mathcal{K}_x$ be the set of long and crossing tasks. We say that two tasks k and ℓ are *compatible*, if $\tau_k \cap \tau_\ell = \emptyset$. A *snapshot* of $\mathcal{J}_x$ is a set of compatible tasks of $\mathcal{J}_x$. A *relative schedule* of $\mathcal{J}_x$ is a sequence of snapshots of $\mathcal{J}_x$ such that each task $j \in \mathcal{J}_x$ occurs in a subsequence of consecutive snapshots and any two consecutive snapshots are different. Roughly speaking, each relative schedule corresponds to an order of processing of the long and crossing tasks. Furthermore, to any any given non-preemptive schedule of $\mathcal{J}_x$, one can associate a relative schedule in a natural way by looking at every instant where a task of $\mathcal{J}_x$ starts and ends, and writing the set of tasks of $\mathcal{J}_x$ being processed right after that transition. Then, one can see that the number of snapshots in a relative schedule and the number of different relative schedules of $\mathcal{J}_x$ are bounded by constants depending only on m and the number $k_x(m, \varepsilon)$ of long tasks in $\mathcal{L}_x$.

For a relative schedule of $\mathcal{J}_x$, assuming that the short tasks of $\mathcal{S}_x$ can be preempted, one can formulate the linear program which finds the minimum makespan schedule of $\mathcal{S}_x \cup \mathcal{J}_x$ where the tasks of $\mathcal{J}_x$ are scheduled with respect to the relative schedule and the tasks of $\mathcal{S}_x$ are scheduled in the preemptive greedy manner on the free snapshot processors. Furthermore, this schedule can be turned to a non-preemptive one by accommodating the preempted short tasks of $\mathcal{S}_x$ inside the snapshots. One needs only to increase the corresponding snapshot lengths by small amounts (see Figure 2).

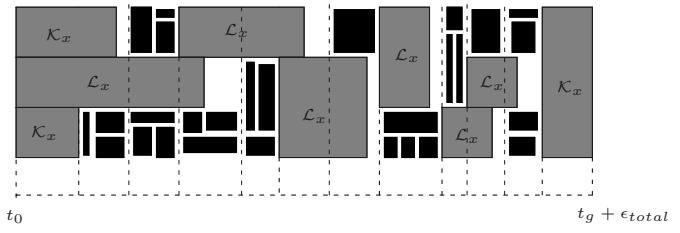


Fig. 2. Deducing a schedule

Then, the total enlargement ε_{total} in the final non-preemptive schedule can be estimated by the total processing time of $qk_x(m,\varepsilon)$ longest tasks of $\mathcal{S}_x$, where $q := 3mB(m)$ and $B(m) \leq m!$ is the mth Bell number. Hence, it is possible to select $k_x(m,\varepsilon)$ such that ε_{total} is kept small enough [3].

Lemma 3. *For any interval I_x, there is a number $k_x(m,\varepsilon)$ with $1 \leq k_x \leq (1+q)^{\frac{m}{\varepsilon^2}}$ such that the total processing time $p_{k_x}+\ldots+p_{(q+1)k_x-1}$ of the $qk_x(m,\varepsilon)$ longest tasks of $\mathcal{S}_x$ is less than $\varepsilon^2 I_x$, where $q := 3mB(m) \leq (m+1)^{m+2}$.*

Using the above result, one can state the final algorithm which finds a feasible schedule of $\mathcal{W}_x$ inside the interval I_{x+1} with respect to the schedule of $\mathcal{K}_x$ in $\sigma'(I)$ as follows: **Step 1.** Compute the set $\mathcal{L}_x$ by finding $k_x(m,\varepsilon)$; **Step 2.** List all possible feasible relative schedules of $\mathcal{J}_x$ and for each such relative schedule deduce the corresponding non-preemptive schedules of $\mathcal{S}_x \cup \mathcal{J}_x$; **Step 3.** Output the schedule with the smallest makespan.

3.2 Tiny versus Short

Here we define $q^*(m,\varepsilon) := ((m+1)^{m+2}+1)^{\frac{m}{\varepsilon^2}+1}$. By Lemma 1 assume that any tiny task is non-crossing in $\sigma(I)$. Then, let $\mathcal{Y}_x \subseteq \mathcal{W}_x$ be the corresponding set of tiny tasks that are non-crossing in I_x. Then, the following holds:

Lemma 4. *The number k_x of long tasks in $\mathcal{L}_x$ is not larger than the number of non-crossing tasks in I_x with $p_j \geq \frac{\varepsilon^2 I_x}{q^*}$. Hence, all tiny tasks are short, i.e. $\mathcal{Y}_x \subseteq \mathcal{S}_x$.*

Proof. First, assume that the number of tasks (scheduled completely in I_x) with length $\geq \frac{\varepsilon^2 I_x}{q^*}$ is larger than $(1+q)^{\frac{m}{\varepsilon^2}}$. In this case, the number of huge tasks in interval I_x is more than $(1+q)^{\frac{m}{\varepsilon^2}}$. By Lemma 3, there exists a $k_x \leq (1+q)^{\frac{m}{\varepsilon^2}}$ such that $p_{k_x}+\ldots+p_{(q+1)k_x-1} \leq \varepsilon^2 I_x$. Then, k_x is less than the number of non-crossing huge tasks in I_x.

Assume now that the number of tasks with length $\geq \frac{\varepsilon^2 I_x}{q^*}$ is at most $(1+q)^{\frac{m}{\varepsilon^2}}$. In this case, we set k_x equal to the number of tasks with length $\geq \frac{\varepsilon^2 I_x}{q^*}$. By the bound on ε_{total}, we obtain that the enlargement of the schedule is at most

$$p_{k_x}+\ldots+p_{(q+1)k_x-1} \leq qk_x p_{k_x} \leq qk_x \frac{\varepsilon^2 I_x}{q^*} < \varepsilon^2 I_x,$$

since $qk_x < q^*(m,\varepsilon) = ((m+1)^{m+2}+1)^{\frac{m}{\varepsilon^2}+1}$. This implies also that k_x is bounded by the number of non-crossing huge tasks. □

By combining the previously obtained results we derive a lemma that allows us to vary tiny tasks. Again let $\mathcal{Y}_x$ be the set of tiny tasks in I_x corresponding to schedule $\sigma(I)$.

Lemma 5. *Let $\tilde{\mathcal{Y}}_x$ be another set of tiny tasks such that for each $\tau \in 2^M$ $p(\mathcal{Y}_x^\tau) = 0$ implies $p(\tilde{\mathcal{Y}}_x^\tau) = 0$, and $p(\mathcal{Y}_x^\tau) \neq 0$ implies either $|p(\mathcal{Y}_x^\tau) - p(\tilde{\mathcal{Y}}_x^\tau)| \leq \frac{\varepsilon I_x}{2^{m+1}}$ or $p(\tilde{\mathcal{Y}}_x^\tau)=0$. Then, we can generate outputs a feasible schedule of $W_x := (\mathcal{W}_x \setminus \mathcal{Y}_x) \cup \tilde{\mathcal{Y}}_x$ inside I_{x+1} with respect to the schedule of $\mathcal{K}_x$ in $\sigma'(I)$.*

3.3 Scheduling Tiny Tasks and Compact Instances

Now consider the problem of placing tiny tasks in the schedule. In the following we restrict ourselves to the case of scheduling tiny tasks of the same type, say τ. We say that two tiny tasks $k, \ell \in \mathcal{TT}^\tau$ with $\frac{p_k}{w_k} < \frac{p_\ell}{w_\ell}$ are scheduled by *Smith's rule* in a schedule σ if it holds either $z(k) \le z(\ell)$, or $z(\ell) < x(k)$. In other words, if the two tasks are available at the same interval (that means ℓ is not completed before k is released), then the task k of smaller value p_k/w_k is scheduled first with respect to intervals. Note that the tasks of $\mathcal{TT}^\tau_x$ have to be scheduled in the order of increasing values p_j/w_j [16].

Lemma 6. *For a feasible schedule σ there is an ε-schedule σ' such that for any type $\tau \in 2^M$ the tiny tasks of $\mathcal{TT}^\tau$ are scheduled by Smith's rule in σ'.*

Proof. W.l.o.g. let I_1, I_L be the first and the last interval in a feasible schedule σ. By Lemma 1, assume that all the tasks of $\mathcal{TT}^\tau$ are non-crossing in σ and by Lemma 2 all quotients p_j/w_j are different. Let $\mathcal{Y}^\tau_x$ be the set of tiny tasks of type τ that are scheduled in interval I_x and let $p(\mathcal{Y}^\tau_x)$ be the total processing time of these tasks in I_x. Consider the following $LP(\tau)$:

$$\begin{array}{lll} \text{Minimize} & & \sum_{j\in\mathcal{TT}^\tau} w_j \sum_{x=x(j)}^{L} y_{j,x} R_x \\ \text{s.t.} & (1) & \sum_{x=x(j)}^{L} y_{j,x} \;=\; 1, \quad \forall j \in \mathcal{TT}^\tau, \\ & (2) & \sum_{j\in\mathcal{TT}^\tau,\; x(j)\,\le\, x} y_{j,x}\, p_j \le p(\mathcal{Y}^\tau_x), \quad \forall I_x, \\ & (3) & y_{j,x} \;\ge\; 0, \quad \forall T_j \in \mathcal{TT}^\tau,\; x = x(j), \ldots, L. \end{array}$$

First, the objective value of the linear program is not larger than the weighted average completion time for the tiny tasks in σ. In other words, the value of an optimal fractional solution is a lower bound of the weighted completion time $\sum_{j\in\mathcal{TT}^\tau} w_j C_j$. Consider an optimal solution $(y^*_{j,x})$ of the linear program. Suppose that there are two tasks ℓ and k that are not scheduled by Smith's rule. W.l.o.g. we can consider the case when $y^*_{\ell,x_\ell} > 0$, $y^*_{k,x_k} > 0$, $x(k) \le x(\ell) \le x_\ell < x_k$ and $\frac{p_k}{w_k} < \frac{p_\ell}{w_\ell}$.

Then, there exist values z_ℓ and z_k such that $0 < z_\ell \le y^*_{\ell,x_\ell}$ and $0 < z_k \le y^*_{k,x_k}$ and $z_\ell p_\ell = z_k p_k$. Now we exchange parts of the variables:

$$\begin{array}{ll} y'_{\ell,x_\ell} = y^*_{\ell,x_\ell} - z_\ell & y'_{\ell,x_k} = y^*_{\ell,x_k} + z_\ell \\ y'_{k,x_k} = y^*_{k,x_k} - z_k & y'_{k,x_\ell} = y^*_{k,x_\ell} + z_k \end{array}$$

and $y'_{j,x} = y^*_{j,x}$ for the remaining variables.

The new solution $(y'_{j,x})$ is feasible and the objective value $\sum_{T_j\in\mathcal{TT}^\tau} w_j \sum_{x=x(j)}^{L} y'_{j,x} R_x$ is equal to $\sum_{T_j\in\mathcal{TT}^\tau} w_j \sum_{x=x(j)}^{L} y^*_{j,x} R_x + R_{\ell,k}$, where $R_{\ell,k} = (R_{x_\ell} - R_{x_k})(w_k z_k - z_\ell w_\ell)$. By $z_k = z_\ell \frac{p_\ell}{p_k}$, the second factor $(w_k z_k - z_\ell w_\ell) = z_\ell(w_k \frac{p_\ell}{p_k} - w_\ell)$. Since $\frac{p_k}{w_k} < \frac{p_\ell}{w_\ell}$ and $z_\ell > 0$, the second factor is larger than 0. The inequality $x_\ell < x_k$ implies $R_{x_\ell} < R_{x_k}$ and $R_{\ell,k} < 0$. In other words, the new solution $(y'_{j,x})$ has a lower objective value and gives us a contradiction.

Now we use some properties of the above linear program. There is an optimal fractional solution such that for each interval I_x there is at most one tiny task

j_x of type τ with $x_{j,x} \in (0,1)$ and that is assigned for the first time (one can apply Smith's rule in a greedy manner). To turn such a fractional solution into an integral assignment of tiny tasks to intervals, one needs only to increase the values $p(\mathcal{Y}_x^\tau)$ by $\frac{\varepsilon^2 I_x}{q^*}$. Then, each task j_x fits completely into I_x. It is not hard to see that the results of Lemma 5 can be used. Thus, we apply the shifting* to σ, replace the sets $\mathcal{Y}_x^\tau$ by the sets that correspond to the integral assignments. The final ε-schedule σ' is constructed. □

By using the above Lemma and the ideas from [1] we get:

Lemma 7. *With $1 + O(\varepsilon)$ loss and in $O(n \log n)$ time, one can transform an instance I into instance I' such that for an interval I_x and a type $\tau \in 2^M$ the total number of tasks of type τ released at I_x is bounded by a constant, i.e. $|\mathcal{TT}(I')_x^\tau \cup \mathcal{HT}(I')_x^\tau| \leq h^*(m, \varepsilon)$.*

4 The Dynamic Programming Algorithm

As we mentioned it above, we will use a dynamic program to compute approximate solutions for our scheduling problem. To be able to formulate this dynamic program and show that the algorithm solving it has the desired running time, we need only one final result:

Lemma 8. *For any feasible schedule σ, there is an ε-schedule σ' of σ such that in σ' the tasks released in an interval I_x are scheduled within a constant number $d^*(m, \varepsilon)$ of intervals following I_x.*

Proof. First, partition the time line into a sequence of *superblocks* $\mathcal{SB}_1, \mathcal{SB}_2, \ldots$ such that each superblock $\mathcal{SB}_i = \{I_{s(i)}, \ldots, I_{s'(i)}\}$ consists of $\delta(m, \varepsilon) := s'(i) - s(i) + 1 = \frac{2s^*(\varepsilon)}{\varepsilon}$ consecutive intervals, where $s^*(\varepsilon) = \log_{1+\varepsilon}(1 + \frac{1}{\varepsilon})$. Next, partition each superblock $\mathcal{SB}_i$ into $1/\varepsilon$ *blocks* $\mathcal{B}_{i,1}, \ldots, \mathcal{B}_{i,\frac{1}{\varepsilon}}$ such that each block $\mathcal{B}_{i,\ell}$, $\ell = 1, \ldots, 1/\varepsilon$ consists of $2s^*(\varepsilon)$ consecutive intervals. For a set A of intervals and a schedule σ, let $W(A, \sigma) = \sum_{I_z(j) \in A} w_j R_{z(j)}$ be the total weighted completion time of the tasks that complete in the intervals of A. Since for a superblock $\mathcal{SB}_i$ it holds that $W(\mathcal{SB}_i, \sigma) = \sum_{\ell=1}^{\frac{1}{\varepsilon}} W(\mathcal{B}_{i,\ell}, \sigma)$, there is a block $\mathcal{B}_{i,\bar{\ell}}$ such that $W(\mathcal{B}_{i,\bar{\ell}}, \sigma) \leq \varepsilon W(\mathcal{SB}_i, \sigma)$.

For simplicity, let $b(i)$ and $b'(i) = b(i) + 2s^*(\varepsilon) - 1$ be the indices of the first and last intervals of block $\mathcal{B}_{i,\bar{\ell}}$, let also $t_i := I_{b(i)}/\varepsilon^2$. Then, we partition the set $\mathcal{T}_i := \{j \in T \mid I_{z(j)} \in \mathcal{B}_{i,\bar{\ell}}\}$ of tasks completing in the intervals of block $\mathcal{B}_{i,\bar{\ell}}$ into two subsets $\mathcal{T}_i^-$ and $\mathcal{T}_i^+$, where $\mathcal{T}_i^-$ and $\mathcal{T}_i^+$ contain tasks with $C_j \leq R_{b(i)} + t_i$ and $C_j > R_{b(i)} + t_i$, respectively. Note that $R_{b(i)} + t_i = R_{b(i)} + \frac{I_{b(i)}}{\varepsilon^2} = R_{b(i)}(1 + \frac{1}{\varepsilon}) = R_{b(i)+s^*(\varepsilon)}$.

Next, we apply shifting** to σ and obtain σ'. In σ', the tasks of $\mathcal{T}_i$ run in the intervals $I_{b(i)+1}, \ldots, I_{b'(i)+1}$. In addition, in σ' each task $j \in T$ has at least $\varepsilon(R_{z(j)} - R_{y(j)})$ idle time on the processors of τ_j. Now we construct an ε-schedule σ'' of σ' by rescheduling the tasks of $\mathcal{T}_i$ for each superblock $\mathcal{SB}_i$ as follows: Reschedule the tasks of $\mathcal{T}_i^-$ by eliminating all created idle time from $R_{b(i)+1}$ up to $R_{b(i)+1} + t_i$ (i.e. tasks are shifted backwards); Reschedule the tasks of $\mathcal{T}_i^+$ by

eliminating all created idle time from $R_{b(i)+1} + t_i$ up to $R_{b'(i)+2}$ (i.e. tasks are shifted forward). Thus, the idle times within the intervals $I_{b(i)+1}, \ldots, I_{b'(i)+1}$ are moved to the time point $R_{b(i)+1} + t_i$ (see Figure 3).

Since in any schedule, tasks start and complete consecutively, and $s^*(\varepsilon) = \log_{1+\varepsilon}(1+\frac{1}{\varepsilon})$, the total idle time for intervals $I_{b(i)+1}, \ldots, I_{b'(i)+1}$ in σ' is at least $\sum_{\ell=\ b(i)}^{b'(i)-1} \varepsilon I_\ell = \varepsilon \left(R_{b(i)+2s^*(\varepsilon)} - R_{b(i)}\right) \geq I_{b(i)}((1+1/\varepsilon)^2 - 1) \geq I_{b(i)}/\varepsilon^2$. Then, since each task crosses at most $s^*(\varepsilon)$ intervals, it is not hard to see that in σ'', no task from $\mathcal{T}_i^-$ crosses the time point $\bar{t}_i = R_{b(i)+1} + t_i$. To see that no task from $\mathcal{T}_i^+$ crosses $\bar{t}_i$, observe that the distance between any task in $\mathcal{T}_i^-$ and any task in $\mathcal{T}_i^+$, on any machine, has to be at least $I_{b(i)}/\varepsilon^2$. Therefore, since in σ'' all tasks from $\mathcal{T}_i^-$ complete before $\bar{t}_i = R_{b(i)+1} + t_i = R_{b(i)+1} + \frac{I_{b(i)}}{\varepsilon^2}$, no task is processed on any machine at time $\bar{t}_i$.

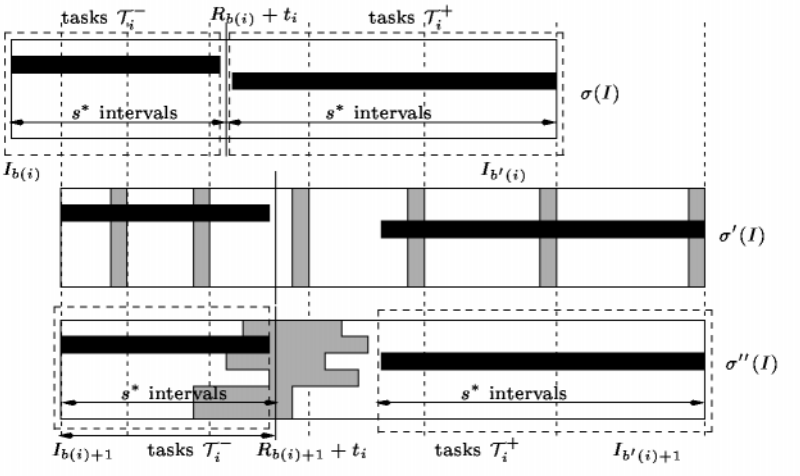


Fig. 3. Creating an idle time

Hence we have shown that in σ'', for each superblock $\mathcal{SB}_i$, there exists at least one time point when all processors are idle. To see that the schedule σ'' is an ε-schedule of σ', one has to consider the tasks of $\mathcal{T}_i^+$, since only their completion times can increase dramatically. For each task $j \in \mathcal{T}_i^+$ the following holds:

$$\begin{aligned} C''_j &\leq C'_j + \varepsilon(R_{b(i)+2s^*(\varepsilon)+1} - R_{b(i)}) = C'_j + \varepsilon R_{b(i)}((1+\varepsilon)^{2s^*(\varepsilon)+1} - 1) \\ &= C'_j + R_{b(i)}\varepsilon\left(\left(1+\tfrac{1}{\varepsilon}\right)^2(1+\varepsilon) - 1\right) \leq C'_j + \varepsilon R_{b(i)}\left(2\left(1+\tfrac{1}{\varepsilon}\right)^2 - 1\right) \\ &\leq C'_j + R_{b(i)}\left(5 + \tfrac{2}{\varepsilon}\right) \leq C'_j + 5R_{b(i)}\left(1+\tfrac{1}{\varepsilon}\right) = C'_j + 5t_i \leq 6C'_j. \end{aligned}$$

We give now the main ideas how the proof can be completed. By Lemmas 1,7 for any interval I_x the total released work can be estimated by $\frac{h^*(m,\varepsilon)}{\varepsilon^3} I_x$. Then

$$\frac{h^*(m,\varepsilon)}{\varepsilon^3} I_x \;=\; \frac{h^*(m,\varepsilon)}{\varepsilon^3(1+\varepsilon)^{c^*(m,\varepsilon)}} I_{x+c^*(m,\varepsilon)} \;\leq\; \frac{\varepsilon^2}{2^{m+1}} I_{x+c^*(m,\varepsilon)},$$

where $c^*(m,\varepsilon)$ is some constant. Hence for any interval I_y, the total work released earlier than $I_{y-c^*(m,\varepsilon)}$ can be bounded by

$$\begin{aligned}\sum_{x\leq y-c^*(m,\varepsilon)} \tfrac{h^*(m,\varepsilon)}{\varepsilon^3} I_x \leq\; & \sum_{x\leq y-c^*(m,\varepsilon)} \tfrac{\varepsilon^2}{2^{m+1}} I_{x+c^*(m,\varepsilon)} \;\leq\; \sum_{t\geq 0} \tfrac{\varepsilon^2 I_y}{2^{m+1}(1+\varepsilon)^t} \\ \leq\; & \tfrac{\varepsilon^2 I_y}{2^{m+1}} \sum_{t\geq 0} \tfrac{1}{(1+\varepsilon)^t} \;\leq\; \tfrac{\varepsilon(1+\varepsilon) I_y}{2^{m+1}} \;\leq\; \tfrac{\varepsilon}{2^m} I_y.\end{aligned}$$

If there is a gap of all idle processor in an interval I_y of length at least $\frac{\varepsilon}{2^m} I_y$, then all tasks released in intervals I_x, $x \leq y - c^*(m,\varepsilon)$ can be completed in this gap. Since each superblock consist of $2s^*(\varepsilon)/\varepsilon$ intervals, in σ'' within any consecutive $4s^*(\varepsilon)/\varepsilon$ intervals there is an interval I_y containing such a time point with idle processors. By using the shifting** and rearranging techniques, in each such interval I_y, we create a gap of all idle processors of length at least εI_y. Suppose that some tasks released at an interval I_x are not completed within $d^*(m,\varepsilon) := c^*(m,\varepsilon) + \frac{4s^*(\varepsilon)}{\varepsilon}$ intervals following I_x. Then, there is a gap of idle processors within the last $\frac{4s^*(\varepsilon)}{\varepsilon}$ consecutive intervals. Hence, we can complete these tasks in this gap by decreasing their completion times, i.e. by rescheduling them backwards. The final ε-schedule of σ is constructed. □

Now we partition the time line into a sequence of *blocks* $\mathcal{D}_1, \mathcal{D}_2, \ldots$, where each block $\mathcal{D}_i$ consists of $d^*(m,\varepsilon)$ consecutive intervals. By Corollary 1, we have to consider only $O(n)$ relevant blocks. Furthermore, by Lemmas 7,8 there is at most a constant number of tasks released in each block, and all of them have to be completed not later than the end of the next block. To finish the proof of our main result, the last step is to use dynamic programming with blocks as units. Here (due to space limitation) we omit the formal description of the dynamic programming algorithm referring to the full version of our paper.

5 Conclusion

In this paper we have presented a PTAS for $Pm|fix_j, r_j|\sum w_j C_j$. The approach we used has the additional benefit that it provides a general framework for designing approximation algorithms for scheduling multiprocessor task with release dates and the weighted sum of completion time as objective. For instance, by following the same line of ideas as above (along with some straightforward modifications) one can also derive a PTAS for the parallel variant of the problem $Pm|size_j, r_j|\sum w_j C_j$. Furthermore, since the described approach seems to be sufficiently general and powerful, it may also prove to be useful in designing a PTAS for the general multiprocessor scheduling problem $Pm|set_j, r_j|\sum w_j C_j$.

References

1. F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Millis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko, Approximation schemes for minimizing average weighted completion time with release dates, *Proceedings 40th IEEE Symposium on Foundations of Computer Science* (1999), 32-43.
2. F. Afrati, E. Bampis, A. V. Fishkin, K. Jansen, C. Kenyon, Scheduling to minimize the average completion time of dedicated tasks, *Proceedings 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, LNCS 1974, Springer Verlag (2000), 454-464.
3. A. K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, Scheduling independent multiprocessor tasks, *Proceedings 5th European Symposium on Algorithms*, LNCS 1284, Springer Verlag (1997), 1-12.
4. A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir, On chromatic sums and distributed resource allocation, *Information and Computation* 140 (1998), 183-202.
5. A. Bar-Noy and M. M. Halldórsson and G. Kortsarz and R. Salman and H. Shachnai, Sum multicoloring of graphs, *Proceedings 7th European Symposium on Algorithms*, LNCS 1643, Springer Verlag (1999), 390-401.
6. P. Brucker and A. Krämer, Polynomial algorithms for resource constrained and multiprocessor task scheduling problems, *European Journal of Operational Research* 90 (1996), 214-226.
7. X. Cai, C.-Y. Lee, and C.-L. Li, Minimizing total completion time in two-processor task systems with prespecified processor allocation, *Naval Research Logistics* 45 (1998), 231-242.
8. S. Chakrabarti, C. A. Philips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein, Improved scheduling algorithms for minsum criteria, *Proceedings 23rd International Colloquium on Automata, Languages and Programming*, LNCS 1099, Springer Verlag (1996), 646-657.
9. M. Drozdowski, Scheduling multiprocessor tasks - an overview, *European Journal on Operations Research*, 94 (1996), 215-230.
10. U. Feige and J. Kilian, Zero-knowledge and the chromatic number, in *Journal of Computer and System Science* 57(2) (1998), 187-199.
11. M. R. Garey and D. S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, Freeman, San Francisco, CA, 1979.
12. R. L. Graham, E. L. Lawler, J. K. Lenstra, K. Rinnooy Kan, Optimization and approximation in deterministic scheduling: a survey, *Annals of Discrete Mathematics* 5 (1979), 287-326.
13. L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, Scheduling to minimize average time: Offline and online algorithm, *Mathematics of Operation Research* 22 (1997), 513-544.
14. J. A. Hoogeveen, S. L. Van de Velde, and B. Veltman, Complexity of scheduling multiprocessor tasks with prespecified processor allocations, *Discrete Applied Mathematics* 55 (1994), 259-272.
15. M. Skutella and G. J. Woeginger, A PTAS for minimizing the weighted sum of job completion times on parallel machines, *Proceedings 31st ACM Symposium on Theory of Computing* (1999), 400-407.
16. W. E. Smith, Various optimizers for single-stage production, *Naval Research Logistic Quarterly* 3 (1956), 59-66.

On the Approximability of Average Completion Time Scheduling under Precedence Constraints

Gerhard J. Woeginger

Institut für Mathematik, TU Graz, Austria

Abstract. We consider the scheduling problem of minimizing the average weighted job completion time on a single machine under precedence constraints. We show that this problem with arbitrary job weights, the special case of the problem where all job weights are one, and several other special cases of the problem all have the same approximability threshold with respect to polynomial time approximation algorithms. Moreover, for the special case of interval order precedence constraints and for the special case of convex bipartite precedence constraints, we give a polynomial time approximation algorithm with worst case performance guarantee arbitrarily close to the golden ratio $\frac{1}{2}(1+\sqrt{5}) \approx 1.61803$.

1 Introduction

We consider the problem of scheduling n jobs on a single machine. Each job J_j $(j = 1, \ldots, n)$ is specified by its length p_j and its weight w_j, where p_j and w_j are non-negative integers. Each job J_j must be scheduled for p_j units of time, and only one job can be scheduled at any point in time. We only consider *non-preemptive* schedules, in which all p_j time units of job J_j must be scheduled consecutively. Precedence constraints are given by a partial order on the jobs; if J_i precedes J_j in the partial order (denoted by $J_i \to J_j$), then J_i must be processed before J_j can begin its processing. In this situation, job J_i is called a *predecessor* of J_j, and J_j is called a *successor* of J_i. Our goal is to find a schedule which minimizes the sum $\sum_{j=1}^{n} w_j C_j$ where C_j is the time at which job J_j completes in the given schedule. In the standard three-field scheduling notation (see e.g. Graham, Lawler, Lenstra & Rinnooy Kan [5]) this problem is denoted by $1 \,|\, prec \,|\, \sum w_j C_j$, and the special case where $w_j \equiv 1$ is denoted by $1 \,|\, prec \,|\, \sum C_j$. Both problems $1 \,|\, prec \,|\, \sum w_j C_j$ and $1 \,|\, prec \,|\, \sum C_j$ are NP-hard in the strong sense (Lawler [7]; Lenstra & Rinnooy Kan [8]).

A polynomial time *ρ-approximation algorithm* is a polynomial time algorithm that always returns a near-optimal solution with cost at most a factor ρ above the optimal cost (where $\rho > 1$ is some fixed real number). The value ρ is called the *worst case performance guarantee* of the approximation algorithm. The *approximability threshold* of a minimization problem is the infimum of all values ρ for which the problem possesses a polynomial time ρ-approximation algorithm.

There are several different polynomial time 2-approximation algorithms known for the problem $1 \,|\, prec \,|\, \sum w_j C_j$. Hall, Schulz, Shmoys & Wein [6] give

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 887–897, 2001.

such a 2-approximation algorithm by using linear programming relaxations. Chudak & Hochbaum [2] design another 2-approximation algorithm that is based on a half integral linear programming relaxation and on a minimum cut computation in an underlying network. Independently of each other, Margot, Queyranne & Wang [9] and Chekuri & Motwani [1] provide (identical) extremely simple, combinatorial polynomial time 2-approximation algorithms. It is an outstanding open problem to determine the exact approximability threshold of $1 \,|\, prec \,|\, \sum w_j C_j$; see e.g. Schuurman & Woeginger [12]. According to the current state of the art, this threshold might be any value between 1 (which would mean: the problem has a PTAS) and 2 (which would mean: the currently known approximation algorithms are already best possible).

Interestingly, the special case $1 \,|\, prec \,|\, \sum C_j$ seems to be no easier to approximate than the general case $1 \,|\, prec \,|\, \sum w_j C_j$. The best approximation algorithms known for $1 \,|\, prec \,|\, \sum C_j$ have performance guarantee 2, i.e., exactly the same guarantee as for the general case. Every constructive approach that works for the special case seems to carry over to the general case. In this paper we will show that the cumulative experiences of the research community with these two problems are not just a coincidence:

Theorem 1. *The approximability thresholds of the following eight special cases of the scheduling problem* $1 \,|\, prec \,|\, \sum w_j C_j$ *all coincide:*

(a) The approximability threshold τ_a *of the general problem* $1 \,|\, prec \,|\, \sum w_j C_j$.
(b) The approximability threshold τ_b *of the special case where* $w_j \leq n^2$ *and* $p_j \leq n^2$.
(c) The approximability threshold τ_c *of the special case where* $w_j \equiv 1$.
(d) The approximability threshold τ_d *of the special case where* $p_j \equiv 1$.
(e) The approximability threshold τ_e *of the special case where* $w_j \equiv 1$ *and* $p_j \in \{0, 1\}$.
(f) The approximability threshold τ_f *of the special case where* $w_j \in \{0, 1\}$ *and* $p_j \equiv 1$.
(g) The approximability threshold τ_g *of the special case where every job has either* $p_j = 0$ *and* $w_j = 1$, *or* $p_j = 1$ *and* $w_j = 0$.
(h) The approximability threshold τ_h *of the special case where every job has either* $p_j = 0$ *and* $w_j = 1$, *or* $p_j = 1$ *and* $w_j = 0$, *and where the existence of a precedence constraint* $J_i \rightarrow J_j$ *implies that* $p_i = 1$ *and* $w_i = 0$, *and that* $p_j = 0$ *and* $w_j = 1$.

Chekuri & Motwani [1] design instances of the restricted form (h) to show that the integrality gap of a linear ordering relaxation of Potts [11] is 2. Our Theorem 1 gives some insight why even these highly restricted special cases yield worst possible integrality gaps. Hall & al. [6] observe that the integrality gap of the time-indexed linear programming formulation of Dyer & Wolsey [3] is 2. One problem with this formulation is that its size is proportional to the overall job length, which in general is not polynomially bounded in the input size. Hall & al. [6] find a way around this problem by using an equivalent interval-indexed formulation. Theorem 1 offers another way around this problem, since it shows

that the thresholds for problems (a) and (b) are equivalent with respect to approximation, and in problem (b) the overall job length is polynomially bounded in the input size.

Our second main result is that for specially structured families of precedence constraints, one can get an improved performance guarantee that is arbitrarily close to the golden ratio $\frac{1}{2}(1+\sqrt{5})$.

Theorem 2. *Let $\phi = \frac{1}{2}(1+\sqrt{5}) \approx 1.61803$ be the positive real root of $\phi^2 = \phi+1$. For any $\varepsilon > 0$, there exists a polynomial time $(\phi+\varepsilon)$-approximation algorithm for problem $1\,|\,prec\,|\,\sum w_jC_j$ in case (a) the precedence constraints form an interval order, and in case (b) the precedence constraints form a convex bipartite order.*

The paper is organized as follows. In Section 2 we collect several useful definitions and notations, and we also summarize important tools from the literature. In Section 3 we show that the eight approximability thresholds stated in Theorem 1 are all equal. In Section 4 we first show that a polynomial time algorithm for a certain auxiliary problem implies a polynomial time $(\phi+\varepsilon)$-approximation algorithm for $1\,|\,prec\,|\,\sum w_jC_j$. Then we show that for interval orders and for convex bipartite orders this auxiliary problem indeed can be solved in polynomial time. This will prove Theorem 2.

2 Definitions, Propositions, and Preliminaries

For any instance I of $1\,|\,prec\,|\,\sum w_jC_j$, we denote its optimal objective value by $\textsc{Opt}(I)$, and we write $p(I) = \sum_{j=1}^{n} p_j$ and $w(I) = \sum_{j=1}^{n} w_j$. For any subset S of jobs, let $p(S) = \sum_{J_j \in S} p_j$ and $w(S) = \sum_{J_j \in S} w_j$. A subset S of jobs is called an *initial set* if for every job $J_j \in S$ also all the predecessors of J_j are in S.

Goemans & Williamson [4] provide a nice geometric way of looking at $1\,|\,prec\,|\,\sum w_jC_j$ via a two-dimensional Gantt chart. The Gantt chart is a big rectangle with its four corners at $(0,0)$, $(0,w(I))$, $(p(I),0)$, and $(p(I),w(I))$. The straight line from corner $(0,w(I))$ to corner $(p(I),0)$ is called the *diagonal* of the Gantt chart. Each job J_j is represented by a *job rectangle* of length p_j and height w_j. A schedule $J_{\pi(1)}, J_{\pi(2)}, \ldots, J_{\pi(n)}$ is represented by the following placement of the job rectangles. The rectangle for the first job $J_{\pi(1)}$ is placed such that its upper left corner lies in $(0,w(I))$. For $j = 2,\ldots,n$ the rectangle for $J_{\pi(i)}$ is placed such that its upper left corner coincides with the lower right corner of the rectangle for $J_{\pi(i-1)}$. Clearly, the rectangle for the last job $J_{\pi(n)}$ then has its lower right corner in $(p(I),0)$. It can be seen [4] that the total weighted completion time in the schedule $J_{\pi(1)}, J_{\pi(2)}, \ldots, J_{\pi(n)}$ equals the total area of the job rectangles plus the total area that lies below these rectangles in the Gantt chart. This total area (that equals the objective value) will be called the *covered area* of the schedule.

Margot, Queyranne & Wang [9] and Chekuri & Motwani [1] show that within polynomial time, any instance I of $1\,|\,prec\,|\,\sum w_jC_j$ can be split into $k \geq 1$ subinstances $I_1,\ldots,I_k$ such that the following two conditions are fulfilled. (i) In any subinstance I_i all initial sets S satisfy $p(S)\,w(I_i) \geq p(I_i)\,w(S)$. (ii) If

σ_i $(i = 1, \ldots, k)$ is an arbitrary optimal schedule for instance I_i, then putting the schedules $\sigma_1, \sigma_2, \ldots, \sigma_k$ in series yields an optimal schedule for the original instance I. With this, any ρ-approximation algorithm for instances that satisfy condition (i) immediately yields a ρ-approximation algorithm for general instances; simply compute ρ-approximate solutions for the instances $I_1, \ldots, I_k$ and put them in series. Moreover, condition (i) implies that in the Gantt chart of any feasible schedule for instance I_i, the lower right corners of all job rectangles lie on or above the diagonal of the chart; hence, the covered area of such a schedule is at least $w(I_i)\, p(I_i)/2$.

Proposition 3. *(Margot, Queyranne & Wang [9] and Chekuri & Motwani [1]) With respect to approximation algorithms for $1 \,|\, prec \,|\, \sum w_j C_j$, we may restrict ourselves to instances I that satisfy $p(S)\, w(I) \geq p(I)\, w(S)$ for any initial set S. Such instances I a priori satisfy $\textsc{Opt}(I) \geq w(I)\, p(I)/2$.* □

For any instance I of $1 \,|\, prec \,|\, \sum w_j C_j$, we denote by $I^{\#}$ the instance that results from I by replacing every job J_j in I by a new job $J_j^{\#}$ with $w_j^{\#} = p_j$ and $p_j^{\#} = w_j$ and by introducing the precedence constraint $J_j^{\#} \rightarrow J_i^{\#}$ if and only if $J_i \rightarrow J_j$ is in I. In other words, $I^{\#}$ results from I by interchanging job weights and lengths and by reversing the precedence constraints. Instance $I^{\#}$ is called the *reverse instance* of I. Chudak & Hochbaum [2] observe that in case $J_{\pi(1)}, J_{\pi(2)}, \ldots, J_{\pi(n)}$ is a feasible schedule for instance I, then $J^{\#}_{\pi(n)}, \ldots, J^{\#}_{\pi(2)}, J^{\#}_{\pi(1)}$ is a feasible schedule for instance $I^{\#}$. It is easily verified that the objective values of these two schedules are the same.

Proposition 4. *(Chudak & Hochbaum [2])*
There is a polynomial time computable one-to-one correspondence between feasible solutions of instances I and $I^{\#}$ that preserves objective values.

3 Equality of the Eight Approximability Thresholds

In this section we will prove Theorem 1. We start by reducing the general case (a) to the polynomially bounded case (b). Consider an arbitrary instance I of $1 \,|\, prec \,|\, \sum w_j C_j$ with $n \geq 16$ jobs. To keep the presentation simple, we will write W for $w(I)$ and P for $p(I)$. By Proposition 3 we may assume without loss of generality that

$$\textsc{Opt}(I) \;\geq\; WP/2. \tag{1}$$

We define another instance I' that results from I by scaling the weights: For every job J_j in I there is a corresponding job J'_j in instance I' with the same processing time $p'_j = p_j$ and with a new weight of

$$w'_j \;=\; \max\{\lceil w_j n^2/W \rceil,\; 1\} \;\leq\; n^2. \tag{2}$$

Note that $w'_j \geq w_j n^2/W$ and that $w'_j \leq w_j n^2/W + 1$. The precedence constraints in I and I' are exactly the same, i.e., $J'_i \rightarrow J'_j$ in I' if and only if $J_i \rightarrow J_j$ in I. We now assume that for some $\rho \leq 2$ we have a ρ-approximate schedule α for

instance I' with objective value $A(I') \leq \rho\,\textsc{Opt}(I')$ and with job completion times $C'_{\alpha(1)}, \ldots, C'_{\alpha(n)}$. If we use α as an approximate schedule for the unscaled instance I with objective value $A(I)$, then corresponding jobs in the two schedules have the same completion times. This yields

$$A(I') \;=\; \sum_{j=1}^{n} w'_{\alpha(j)} C'_{\alpha(j)} \;\geq\; \frac{n^2}{W} \sum_{j=1}^{n} w_{\alpha(j)} C'_{\alpha(j)} \;=\; \frac{n^2}{W}\, A(I). \tag{3}$$

Next we claim that

$$\textsc{Opt}(I') \;\geq\; n \log_2 n \cdot P. \tag{4}$$

Suppose for the sake of contradiction that $\textsc{Opt}(I') < n \log_2 n \cdot P$. Then the inequality in (3), the fact that $A(I') \leq \rho\,\textsc{Opt}(I')$, and the inequalities $\rho \leq 2$ and $n \geq 16$ together yield that

$$\textsc{Opt}(I) \;\leq\; A(I) \;\leq\; \frac{W}{n^2} A(I') \;\leq\; \frac{W}{n^2} \rho \textsc{Opt}(I') \;<\; \frac{W}{n^2} \rho n \log_2 n \cdot P \;\leq\; WP/2. \tag{5}$$

This blatantly contradicts (1) and thus proves (4). Next, we assume without loss of generality that $J_1, J_2, \ldots, J_n$ is an optimal schedule for instance I with job completion times $C_1, \ldots, C_n$. Then

$$\begin{aligned} \textsc{Opt}(I) = \sum_{j=1}^{n} w_j C_j \;&\geq\; \sum_{j=1}^{n} W(w'_j - 1) C_j / n^2 \;=\; \frac{W}{n^2} \left(\sum_{j=1}^{n} w'_j C_j - \sum_{j=1}^{n} C_j \right) \\ &\geq \frac{W}{n^2} \left(\textsc{Opt}(I') - nP \right) \;\geq\; \frac{W}{n^2} \textsc{Opt}(I') \left(1 - 1/\log_2 n\right). \end{aligned} \tag{6}$$

Here we first used that $w'_j \leq w_j n^2 / W + 1$, then that $C_j \leq P$, and finally the inequality in (4). By combining (3), (6), and $A(I') \leq \rho\,\textsc{Opt}(I')$ we conclude that

$$\begin{aligned} A(I)/\textsc{Opt}(I) &\leq \left(W \cdot A(I')/n^2\right) / \left(W \cdot \textsc{Opt}(I')(1 - 1/\log_2 n)/n^2\right) \\ &= A(I') / \left(\textsc{Opt}(I')(1 - 1/\log_2 n)\right) \;\leq\; \rho/(1 - 1/\log_2 n). \end{aligned} \tag{7}$$

Lemma 5. *Assume that there is a polynomial time ρ-approximation algorithm A for the special case (b) of $1\,|\,prec\,|\,\sum w_j C_j$ where $1 \leq w_j \leq n^2$ and $1 \leq p_j \leq n^2$ holds for all jobs J_j. Then for every $\varepsilon > 0$ there exists a polynomial time $(\rho+\varepsilon)$-approximation algorithm for the general problem $1\,|\,prec\,|\,\sum w_j C_j$.*

Proof. Consider an arbitrary instance I of $1\,|\,prec\,|\,\sum w_j C_j$ with n jobs. If $n < 2^{4\rho/\varepsilon}$, then the problem is of constant size and we may solve it in constant time by complete enumeration. Otherwise, $n \geq 2^{4\rho/\varepsilon}$ and $\rho/(1 - 1/\log_2 n)^2 \leq \rho + \varepsilon$. Let I' result from I by scaling the job weights as described above; then by (2) $1 \leq w_j \leq n^2$ holds for all jobs in I'. Let $I'^{\#}$ be the reverse instance of I'; then $1 \leq p_j \leq n^2$ holds for all jobs in $I'^{\#}$. Let $I'^{\#\prime}$ result from $I'^{\#}$ by scaling the job weights as described above; then $1 \leq w_j \leq n^2$ and $1 \leq p_j \leq n^2$ hold for all

jobs in $I'^{\#\prime}$. Note that the three instances I', $I'^{\#}$, and $I'^{\#\prime}$ can be determined in polynomial time.

We apply the polynomial time ρ-approximation algorithm A to instance $I'^{\#\prime}$, and interpret the resulting approximate schedule $\alpha'^{\#\prime}$ as a schedule for $I'^{\#}$. By (7) this yields an approximate schedule $\alpha'^{\#}$ for $I'^{\#}$ with objective value at most $\rho/(1-1/\log_2 n)$ above $\textsc{Opt}(I'^{\#})$. By Proposition 4, the approximate schedule $\alpha'^{\#}$ can be translated into an approximate schedule α' of the same approximation quality for instance I'. Finally, we interpret the approximate schedule α' as a schedule for I. By applying once again (7), we get that the resulting approximate objective value for I is at most $\rho/(1-1/\log_2 n)^2 \leq \rho+\varepsilon$ above $\textsc{Opt}(I)$. This yields the desired polynomial time $(\rho+\varepsilon)$-approximation algorithm for the general problem $1\,|\,prec\,|\,\sum w_jC_j$. □

Our next goal is to reduce the polynomially bounded case (b) in Theorem 1 to the case (c) with unit weights. Consider an arbitrary instance I of $1\,|\,prec\,|\,\sum w_jC_j$ with n jobs where $1 \leq w_j \leq n^2$ and $1 \leq p_j \leq n^2$ holds for all jobs J_j, exactly as described in the statement of Lemma 5. We define another instance I^+ that results from I by splitting every job into several new jobs: For every job J_j with length p_j and weight w_j in instance I, there are w_j corresponding jobs in I^+ that all have weight 1. The first corresponding job K_j^+ has length p_j. The remaining w_j-1 jobs corresponding to J_j all have length 0; this whole group of w_j-1 jobs is denoted by G_j^+. The precedence constraints in I^+ are defined as follows. $K_i^+ \rightarrow K_j^+$ in I^+ if and only if $J_i \rightarrow J_j$ in I. All jobs in G_j^+ have the same set of predecessors: The job K_j^+, together with all jobs in I^+ that correspond to predecessors of J_j in I. All jobs in G_j^+ have the same set of successors: All jobs in I^+ that correspond to successors of J_j in I. This completes the description of instance I^+. Note that I^+ only contains $O(n^3)$ jobs and that it can be computed in polynomial time.

In any 'reasonable' schedule for I^+, all jobs in G_j^+ will be processed right after job K_j^+ and thus will form a contiguous block together with job K_j^+. Otherwise, one could decrease the objective value by moving the jobs in G_j^+ (that all have length zero) directly after job K_j^+. This yields a straightforward one-to-one correspondence between feasible schedules for I and reasonable feasible schedules for I^+: Jobs J_j in a schedule for I may be replaced by their corresponding blocks K_j^+ and G_j^+, and vice versa. Then in I there is one job of weight w_j that completes at a certain time t, whereas in I^+ there are w_j corresponding jobs of weight 1 that all complete at time t. Clearly, this correspondence preserves objective values, and it also is computable in polynomial time.

Lemma 6. *Assume that there is a polynomial time ρ-approximation algorithm A for the special case (e) of $1\,|\,prec\,|\,\sum w_jC_j$ where $w_j \equiv 1$ and $p_j \in \{0,1\}$. Then there also exists a polynomial time ρ-approximation algorithm for the special case (b) of $1\,|\,prec\,|\,\sum w_jC_j$ where $1 \leq w_j \leq n^2$ and $1 \leq p_j \leq n^2$ holds for all jobs.*

Proof. Consider an arbitrary instance I of $1\,|\,prec\,|\,\sum w_jC_j$ with n jobs where $1 \leq w_j \leq n^2$ and $1 \leq p_j \leq n^2$ holds. Let I^+ result from I by splitting the jobs

as described above; then $w_j \equiv 1$ in I^+. Let $I^{+\#}$ be the reverse instance of I^+; then $p_j \equiv 1$ in $I^{+\#}$. Let $I^{+\#+}$ result from $I^{+\#}$ by splitting the jobs as described above; then $w_j \equiv 1$ and $p_j \in \{0,1\}$ in $I^{+\#+}$. Note that I^+, $I^{+\#}$, and $I^{+\#+}$ all have the same optimal objective value and all can be determined in polynomial time.

We apply the polynomial time ρ-approximation algorithm A to instance $I^{+\#+}$. We interprete the resulting approximate schedule as a schedule for $I^{+\#}$, translate it into an approximate schedule for I^+ by applying Proposition 4, and interprete the resulting schedule as an approximate schedule for I. By the above discussion and by Proposition 4, all these translations do not change the objective value and thus yield a ρ-approximation for I. □

Lemma 7. *Assume that there is a polynomial time ρ-approximation algorithm A for the special case (g) of $1\,|\,prec\,|\,\sum w_jC_j$ where every job has either $p_j = 0$ and $w_j = 1$, or $p_j = 1$ and $w_j = 0$. Then there also exists a polynomial time ρ-approximation algorithm for the special case (e) of $1\,|\,prec\,|\,\sum w_jC_j$ where $w_j \equiv 1$ and $p_j \in \{0,1\}$.*

Proof. Omitted in this extended abstract. □

Lemma 8. *Assume that there is a polynomial time ρ-approximation algorithm A for the special case (h) of $1\,|\,prec\,|\,\sum w_jC_j$ where every job has either $p_j = 0$ and $w_j = 1$, or $p_j = 1$ and $w_j = 0$, and where the existence of a precedence constraint $J_i \to J_j$ implies that $p_i = 1$ and $w_i = 0$, and that $p_j = 0$ and $w_j = 1$. Then there also exists a polynomial time ρ-approximation algorithm for the special case (g) of $1\,|\,prec\,|\,\sum w_jC_j$ where every job has either $p_j = 0$ and $w_j = 1$, or $p_j = 1$ and $w_j = 0$.*

Proof. Consider an arbitrary instance I of $1\,|\,prec\,|\,\sum w_jC_j$ where every job has either $p_j = 0$ and $w_j = 1$ (such a job will be called a 0-job), or $p_j = 1$ and $w_j = 0$ (such a job will be called a 1-job). We construct a new instance that results from I by removing all the precedence constraints (i) between 0-jobs, (ii) between 1-jobs, and (iii) from 0-jobs to 1-jobs. The resulting instance I^b has bipartite precedence constraints where the 0-jobs form one class and the 1-jobs form the other class of the bipartition. Therefore, instance I^b is of the type (h).

We will now prove that every feasible schedule for the bipartite instance I^b can be transformed into a feasible schedule for the original instance I without increasing the objective value. Clearly, the statement in the lemma will follow from this.

Hence, consider a feasible schedule for I^b. Renumber the jobs such that this schedule processes the jobs in the ordering $J_1, J_2, \ldots, J_n$ with job completion times $C_1, C_2, \ldots, C_n$. We interprete this schedule as a schedule for the original instance I, and we call a pair of jobs J_i and J_j a *violated* pair, if $J_i \to J_j$ in I and if $i > j$. Consider a violated pair J_i and J_j with the difference $i - j$ as small as possible. Consider an intermediate job J_k with $j < k < i$. If J_k was a predecessor of J_i in I, then $J_k \to J_i \to J_j$, and if J_k was a predecessor of J_j in I, then $J_k \to J_j$. In either case, the jobs J_k and J_j would form another violated

pair with $k-j < i-j$. If J_k was a successor of J_j in I, then $J_i \to J_j \to J_k$, and if J_k was a successor of J_i in I, then $J_i \to J_k$. In either case, the jobs J_i and J_k would form another violated pair with $i-k < i-j$. Hence, all intermediate jobs between J_j and J_i are neither predecessors nor successors of J_i and J_j. We distinguish two cases: (i) J_i is a 0-job. Then we remove J_i from the schedule and reinsert it immediately before J_j. By moving this 0-job to an earlier point in time, we will not increase the objective value of the schedule. (ii) J_i is a 1-job. Since the considered schedule is feasible for I^b, in this case also J_j must be a 1-job. We remove J_j from the schedule and reinsert it immediately after J_i. This increases the completion time of J_j from C_j to C_i, and it decreases the completion time of the $i-j$ jobs $J_{j+1}, \ldots, J_i$ all by 1. Note that $i-j \geq C_i - C_j$ since there must be at least $C_i - C_j$ 1-jobs among $J_{j+1}, \ldots, J_i$. Hence, by moving J_j we will not increase the objective value of the schedule.

To summarize, in either case we resolve the violated pair J_i and J_j, we do not create any new violated pairs, and we end up with another feasible schedule for I^b. By repeating this procedure over and over again, we will eventually get rid of all violated pairs without ever increasing the objective value. The resulting schedule will be feasible for the original instance I, exactly as we desired. □

Proof of Theorem 1. Now let us finally prove our main result. Since problem (e) is a special case of problem (c), since problem (c) is a special case of problem (b), and since (b) is a special case of (a), we have

$$\tau_e \;\leq\; \tau_c \;\leq\; \tau_b \;\leq\; \tau_a.$$

Lemmas 5 and 6 together yield that $\tau_a \leq \tau_b \leq \tau_e$. Therefore, the above four approximability thresholds all coincide. Moreover, Lemma 7 yields $\tau_g \geq \tau_e = \tau_a$ which together with $\tau_a \geq \tau_g$ implies $\tau_g = \tau_a$. Similarly, Lemma 8 yields $\tau_h \geq \tau_g = \tau_a$ which together with $\tau_a \geq \tau_h$ implies $\tau_h = \tau_a$. Finally, the discussion after Proposition 4 yields $\tau_c = \tau_d$ and $\tau_e = \tau_f$. □

4 Nice Precedence Constraints

In this section we will derive a polynomial time $(\phi + \varepsilon)$-approximation for $1 \,|\, prec \,|\, \sum w_j C_j$ for certain 'nice' classes of precedence constraints that include interval orders and convex bipartite orders. This approximation algorithm is based on exact algorithms for the following auxiliary problem.

PROBLEM: GOOD INITIAL SET
INSTANCE: An instance I of $1 \,|\, prec \,|\, \sum w_j C_j$, i.e., a set of precedence constrained jobs J_j $(j = 1, \ldots, n)$ with non-negative integer lengths p_j and non-negative integer weights w_j. A real number γ with $0 < \gamma \leq 1/2$.
QUESTION: Does there exist an initial set T that simultaneously satisfies $p(T) \leq (1/2 + \gamma)p(I)$ and $(1/2 - \gamma)w(I) \leq w(T)$?

Theorem 9. *Let $\mathcal{C}$ be a class of precedence constraints such that the restriction of the GOOD INITIAL SET problem to precedence constraints from class $\mathcal{C}$ is*

solvable in polynomial time. Then for any $\varepsilon > 0$, the restriction of problem $1\,|\,prec\,|\,\sum w_jC_j$ to precedence constraints from class $\mathcal{C}$ has a polynomial time $(\phi+\varepsilon)$-approximation algorithm.

Proof. Consider an instance I of $1\,|\,prec\,|\,\sum w_jC_j$ with precedence constraints from class $\mathcal{C}$. We will write W short for $w(I)$, and P short for $p(I)$. By Proposition 3 we may assume that $p(S)\,W \geq w(S)\,P$ holds for any initial set S in I, and that $\textsc{Opt}(I) \geq WP/2$. Take an arbitrary real ε with $0 < \varepsilon < 1/4$ and define $\ell = \lceil 2\phi/\varepsilon \rceil$. For this choice of ℓ, the following inequality is satisfied for all γ with $0 < \gamma \leq 1/2$:

$$\frac{1}{2} + 2(\gamma - \frac{1}{2\ell})^2 \;\geq\; (\frac{1}{2} + 2\gamma^2) \;/\; (1+\varepsilon/\phi) \tag{8}$$

We call the polynomial time algorithm for problem GOOD INITIAL SET on the inputs I and $\gamma_k = k/2\ell$ for $k = 1, 2, \ldots, \ell$ until we detect a value $\gamma = \gamma_k$ that yields a YES-instance. Let T be the corresponding initial set of jobs, and let U be the set of remaining jobs that are not contained in T. We construct an approximate solution for the scheduling instance I that first runs all the jobs in T in any feasible order, and then runs all the jobs in U in any feasible order. We denote the objective value of this approximate solution by $A(I)$. Clearly, this approximate solution can be determined in polynomial time.

Now consider the two-dimensional Gantt chart for the approximate schedule (see the discussion in Section 2). Since $p(T) \leq (1/2+\gamma)P$, all rectangles for jobs in T lie to the left of the line $x = (1/2+\gamma)P$. Moreover, $w(T) \geq (1/2-\gamma)W$ implies $w(U) \leq (1/2+\gamma)W$, and thus all rectangles for jobs in U lie below the line $y = (1/2+\gamma)W$. To summarize, not a single job rectangle protrudes into the 'forbidden' region that lies to the right of the line $x = (1/2+\gamma)P$ and above the line $y = (1/2+\gamma)W$. Since the area of this forbidden region equals $(1/2-\gamma)^2\,WP$, the remaining area in the chart that may contribute to the objective value of the approximate schedule is at most $WP - (1/2-\gamma)^2\,WP$. This yields

$$A(I) \;\leq\; (3/4 + \gamma - \gamma^2)\,WP. \tag{9}$$

Now consider the two-dimensional Gantt chart for an optimal schedule for instance I. Since we assumed by Proposition 3 that $p(S)\,W \geq w(S)\,P$ holds for any initial set S, in this Gantt chart the whole region below the diagonal must belong to the covered area. Moreover it can be shown that the covered area in the Gantt chart must include the rectangular region $\mathcal{R}$ that lies to the left of the vertical line $x = (1/2+\gamma-1/2\ell)P$, and below the horizontal line $y = (1/2+\gamma-1/2\ell)W$. This yields

$$\textsc{Opt}(I) \;\geq\; WP/2 + 2(\gamma - 1/2\ell)^2\,WP \;\geq\; (1/2 + 2\gamma^2)\,WP \;/\; (1+\varepsilon/\phi). \tag{10}$$

Here the first inequality follows from the area estimation, and the second inequality follows from (8). By combining (9) and (10), we conclude that

$$\frac{A(I)}{\textsc{Opt}(I)} \;\leq\; (1+\varepsilon/\phi)\,\frac{3/4+\gamma-\gamma^2}{1/2+2\gamma^2} \;\leq\; (1+\varepsilon/\phi)\,\phi \;=\; \phi+\varepsilon. \tag{11}$$

The final inequality follows by elementary calculus: On the interval $(0, 1/2]$, the function $f(\gamma) = (3/4 + \gamma - \gamma^2) \;/\; (1/2 + 2\gamma^2)$ takes its maximum value at $\gamma = \frac{1}{2}(\sqrt{5} - 2)$, and this maximum value equals $\phi = \frac{1}{2}(1 + \sqrt{5})$. The proof is complete. □

An *interval order* (see e.g. Möhring [10]) on the jobs $J_1, \ldots, J_n$ is specified by a set of n intervals $\mathcal{I}_1, \ldots, \mathcal{I}_n$ along the real line. Then $J_i \to J_j$ holds if and only if interval $\mathcal{I}_i$ lies completely to the left of interval $\mathcal{I}_j$.

Lemma 10. *Consider the subset of instances I of $1 \,|\, prec \,|\, \sum w_j C_j$ with n jobs for which $w_j \leq n^2$ and $p_j \leq n^2$ holds for all jobs and for which the precedence constraints form an interval order. The restriction of the GOOD INITIAL SET problem to instances (I, γ) where I is from this subset is solvable in polynomial time.*

Proof. Omitted in this extended abstract. □

Proof of Theorem 2(a). By Theorem 9 and Lemma 10, there exists a polynomial time $(\phi+\varepsilon)$-approximation algorithm for the special case of $1 \,|\, prec \,|\, \sum w_j C_j$ where $w_j \leq n^2$ and $p_j \leq n^2$ and where the precedence constraints form an interval order. By Lemma 5, this approximability result carries over to the special case of $1 \,|\, prec \,|\, \sum w_j C_j$ under interval order precedence constraints with *arbitrary* job lengths and job weights. Lemma 5 can be applied, since its proof only reverses the precedence constraints, and since the reverse of an interval order is again an interval order. □

A *bipartite order* (see Möhring [10]) on a set of jobs is defined as follows. The jobs are classified into two types. The *minus-jobs* $J_1^-, \ldots, J_a^-$ do not have any predecessors, and the *plus-jobs* $J_1^+, \ldots, J_b^+$ do not have any successors. The only precedence constraints are of the type $J_i^- \to J_j^+$, that is from minus-jobs to plus-jobs. The class of convex bipartite orders forms a proper subset of the class of general bipartite orders, and it is a proper superset of the class of strong bipartite orders (see Möhring [10]). A bipartite order is a *convex bipartite order* if for every $j = 1, \ldots, b$ there exist two indices $\ell(j)$ and $r(j)$ such that $J_i^- \to J_j^+$ holds if and only if $\ell(j) \leq i \leq r(j)$. In other words, the predecessor set of every plus-job forms an interval within the minus-jobs.

Lemma 11. *Consider the subset of instances I of $1 \,|\, prec \,|\, \sum w_j C_j$ with n jobs for which $w_j \leq n^2$ and $p_j \leq n^2$ holds for all jobs and for which the precedence constraints form a convex bipartite order. The restriction of the GOOD INITIAL SET problem to instances (I, γ) where I is from this subset is solvable in polynomial time.*

Proof. Omitted in this extended abstract. □

Proof of Theorem 2(b). The argument is almost identical to the proof of Theorem 2(a) in the preceding subsection. Theorem 9 and Lemma 11 yield the existence of a polynomial time $(\phi + \varepsilon)$-approximation algorithm for the special

case where $w_j \leq n^2$ and $p_j \leq n^2$, and Lemma 5 can be used to carry this over to $1 \,|\, prec \,|\, \sum w_j C_j$ under convex bipartite precedence constraints with arbitrary job lengths and arbitrary job weights. □

Acknowledgement. This work has been supported by the START program Y43-MAT of the Austrian Ministry of Science.

References

1. C. Chekuri and R. Motwani [1999]. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics 98*, 29–38.
2. F. Chudak and D.S. Hochbaum [1999]. A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Operations Research Letters 25*, 199–204.
3. M.E. Dyer and L.A. Wolsey [1990]. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics 26*, 255–270.
4. M.X. Goemans and D.P. Williamson [2000]. Two-dimensional Gantt charts and a scheduling algorithm of Lawler. *Siam Journal on Discrete Mathematics 13*, 281–294.
5. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan [1979]. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics 5*, 287–326.
6. L.A. Hall, A.S. Schulz, D.B. Shmoys, and J. Wein [1997]. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research 22*, 513–544.
7. E.L. Lawler [1978]. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics 2*, 75–90.
8. J.K. Lenstra and A.H.G. Rinnooy Kan [1978]. Complexity of scheduling under precedence constraints. *Operations Research 26*, 22–35.
9. F. Margot, M. Queyranne, and Y. Wang [1997]. Decompositions, network flows, and a precedence constrained single machine scheduling problem. Report #2000-29, Department of Mathematics, University of Kentucky, Lexington.
10. R.H. Möhring [1989]. Computationally tractable classes of ordered sets. In: I. Rival (ed.) *Algorithms and Order*, Kluwer Academic Publishers, 105–193.
11. C.N. Potts [1980]. An algorithm for the single machine sequencing problem with precedence constraints. *Mathematical Programming Study 13*, 78–87.
12. P. Schuurman and G.J. Woeginger [1999]. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling 2*, 203–213.

Optimistic Asynchronous Multi-party Contract Signing with Reduced Number of Rounds

Birgit Baum-Waidner

Entrust Technologies (Switzerland)
birgit.baum@entrust.com

Abstract. Optimistic asynchronous multi-party contract signing protocols have received attention in recent years as a compromise between efficient protocols and protocols avoiding a third party as a bottleneck of security. "Optimistic" roughly means: in case all participants are honest and receive the messages from the other participants as expected, the third party is not involved at all. The best solutions known so far terminate within $t + 2$ rounds in the optimistic case, for any fixed set of n signatories and allowing up to $t < n$ dishonest signatories. The protocols presented here achieve a major improvement compared to the state of the art: The number of rounds R is reduced from $O(t)$ to $O(1)$ for all $n \geq 2t + 1$, and for $n < 2t + 1$, R grows remarkably slowly compared with numbers of rounds in $O(t)$: If $t \approx \frac{k}{k+1}n$ then $R \approx 2k$.[1]

1 Introduction

A *contract signing protocol* is a protocol that allows n signatories to sign a contract text such that, even if up to t, $t < n$, of them are dishonest, either *all* honest signatories obtain a signed contract, or nobody obtains it [3,9].[2] Dishonest signatories can arbitrarily deviate from their protocol (Byzantine model). We assume an *asynchronous* network, i.e., there are no upper bounds on network delays.

Multi-party contract signing has obvious applications in secure electronic commerce, and is the basis for the solution of many related fairness problems, like multi-party certified mail [2].

By using a *third party*, T, that is always honest, the problem can be trivially solved [19]: T collects a digital signature from each of the n signatories, and either redistributes the n signatures, or aborts the contract in case not all n signatures arrive. Security depends fully on T; therefore most research has been focused on getting rid of T as trust and performance bottleneck.

Unfortunately, one cannot get rid of T completely: For $n = 2$ no deterministic protocol without third party exists [12], and each probabilistic protocol has an error probability at least linear in the number of rounds [8]. Therefore the goal is to minimize the involvement of T as much as possible: *Optimistic* protocols

[1] The complete version including the detailed proofs can be found in [1].

[2] A precise definition of the asynchronous n-party contract signing problem is given in Sect. 3.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 898–911, 2001.

depend on a third party T, but in such a way that T is *not* actively involved in case all signatories are honest and "patient enough;" only for recovery purposes T might become active [4,8,17].

Optimistic contract signing protocols have been first described for *synchronous* networks [3,4,17] which is the stronger and more unrealistic model but allows protocols to manage within 2 normal phases plus 2 recovery phases, for any number of signatories and any fault assumption. 2-party protocols for *asynchronous* networks have been described in [5,13,18]. The n-party case was first investigated in [6]. Independently, but subsequently, protocols for the 3-party and n-party case were proposed in [13] and [14], respectively, requiring $O(n^2)$ rounds for $t = n - 1$. In [7], an optimistic asynchronous n-party contract signing protocol was presented requiring $t + 2$ rounds only, and $t + 4$ in the worst case, which is the best result achieved so far and round-optimal in case of $n = t + 1$ (as shown in [14]).

The protocols presented here achieve a significant improvement in case $t < n - 1$: The number of rounds, R, in the optimistic case is

$$R = \begin{cases} 2, & \text{if } n \geq 2t + 1; \\ 2\lfloor \frac{t+1}{n-t} \rfloor + \min(2, (t+1) \mod (n-t)) & \text{if } t + 2 < n \leq 2t; \\ t + 1, & \text{if } n = t + 2; \\ t + 2, & \text{if } n = t + 1; \end{cases} \quad (1)$$

This means a major improvement compared to the state of the art: R is reduced from $O(t)$ to $O(1)$ for all $n \geq 2t+1$, and for $n \leq 2t$, R grows remarkably slowly compared with numbers of rounds in $O(t)$: If $t \approx \frac{k}{k+1} n$ then $R \approx 2k$.

This enables higher numbers of participants for practical use.

2 Model and Notation

Our model and notation are identical to those in [7]:

Let $P_1, \ldots, P_n$ denote the *signatories,* T a *third party,* and $V_1, \ldots, V_{n'}$ the potential *verifiers* of signed contracts. Each *party* represents a machine that executes a certain protocol and that serves a specific user (e.g., human being or another protocol). Parties can receive local *inputs* from their users (e.g., the command to sign a certain contract) and can generate certain local *outputs* for their users (the result of the protocol, e.g. a message that the contract was signed successfully).

The signatories P_i and third party T are able to *digitally sign* messages, and all parties are able to verify their signatures [15]. The signature on message m associated with P_X (or T) is denoted by $\mathsf{sign}_X(m)$ (or $\mathsf{sign}_T(m)$). In our complexity analyses we assume that $\mathsf{sign}(m)$ has constant length, independent of the length of m (e.g., because m is hashed before signing [16]). We abstract from the error probabilities introduced by cryptographic signature schemes and assume that signatures are unforgeable (the translation into a cryptographic model is straightforward).

Adversary model. We assume that up to t (for a given $t < n$) of the n signatories, and for some requirements also T and all verifiers might be *dishonest.* Dishonest parties can behave arbitrarily and be coordinated by a single party, called the *adversary.* Security properties must always hold for *all* adversaries.

Network. All messages sent to or from T or any V_i are reliably delivered, eventually. Messages between any P_i and P_j might never be delivered. We do not require any particular level of synchronization, nor do we assume that messages are delivered in order. The decision on which of all sent messages to deliver next is taken by the adversary. The adversary can read all messages from, and can insert additional messages into, all channels.

Rounds. In contrast to the "phases" in synchronous networks, the rounds in asynchronous networks are not synchronized. Each message is sent and received in the context of a certain round and will usually be based on messages of previous rounds, or on the local decision no longer to wait for any messages.

All-honest case. This is a special case where we assume that *all* n signatories are honest and *all* messages sent are reliably delivered, eventually.

We make use of a few conventions, in order to simplify presentation:

Local timeouts. If we say that *"X waits for certain messages, but can stop waiting any time"* we mean more precisely the following: X accepts a special input wakeup from its user, for each protocol execution. "Waiting for certain messages" means that X continues to receive messages but does not proceed in the protocol run until either the messages have been received as expected, or wakeup is input.

3 Definitions

Definition 1 (Asynchronous Multi-party Contract Signing).[3] An *Asynchronous Multi-Party Contract Signing Scheme* (asynchronous MPCS) consists of two protocols:

- sign$[P_1, \ldots, P_n]$, for signing a contract with signatories $P_1, \ldots, P_n$. sign[] might involve an additional party, T, in which case we call it an *MPCS with third party.*
- verify$[P_i, V_j]$, to allow P_i to show its contract to any verifier, V_j, $j \in \{1, \ldots, n'\}$. verify[] never involves T, i.e., it is always a 2-party protocol only.

A signatory P_i starts sign[] on its local input $(decision_i, tid_i, terms_i, contr_i)$, $decision_i \in \{\mathsf{sign}, \mathsf{reject}\}$.

tid_i is a transaction identifier which should be unique for each execution of sign[]. $terms_i$ contains information about the protocol used including the number and the identities of the participants and of T, their public keys, and the assumed maximum number t of dishonest signatories. $contr_i$ is the contract text to be

[3] In contrast to [7], we assume that a party not willing to sign might have to be present in the protocol, as this is the case for the protocols presented in Section 4. The protocols presented in Section 6 will not need this presence.

signed. To avoid (unintended or malicious) collisions, we require P_i to refuse concluding a contract by executing a protocol with the same tid_i and $terms_i$ as of one in which P_i is or has already been participating. If necessary, the protocol might be re-run with a new tid_i. This requirement allows protocol executions with identical *tid* and different *terms* to run independently without impacting each other. For practical reasons, this makes sense because the requirement for unique *tid*s can be removed.[4]

P_i may receive a local input $(\mathsf{wakeup}, tid_i, terms_i)$ any time, in order to enforce progress of sign[].

Upon termination sign[] produces a local output $(tid_i, terms_i, contr_i, d_i)$ for P_i, with $d_i \in \{\mathsf{signed}, \mathsf{failed}\}$. We will simply say "$P_i$ decides d_i."

To show the contract, a signatory P_i may start verify[] with verifier V_j on P_i's local input $(\mathsf{show}, V_j, tid_i, terms_i, contr_i)$. A verifier V_j starts verify[] on a local input $(\mathsf{verify}, P_i, tid_V, terms_V, contr_V)$. V_j may receive a local input $(\mathsf{wakeup}, tid_V, terms_V)$ any time. Upon termination $\mathsf{verify}[P_i, V_j]$ produces a local output $(tid_V, terms_V, contr_V, d_V)$ with $d_V \in \{\mathsf{signed}, \mathsf{verify\_failed}\}$ for V_j. We will simply say "V_j decides d_V." No output is produced for P_i.

Note that the local service interfaces do not show protocol information like exchanged messages or information proving a valid contract, although that information is produced by the protocols themselves and serves as the basis for the outputs. This way also exotic solutions are included, e.g., that a proof of a contract cannot necessarily be handled as one piece of information but might require a protocol itself.

The following requirements must be satisfied:

- **(R1)** *Correct execution.* In the all-honest case, if all signatories start with the same input (sign, *tid*, *terms*, *contr*), and no signatory receives (wakeup, *tid*, *terms*), then all signatories terminate and decide signed.
- **(R2)** *Unforgeability.* If an honest P_i never received input (sign, $tid, terms, contr$) then *no* honest V_j that receives input $(\mathsf{verify}, P_{i'}, tid, terms, contr)$, for any $P_{i'}$, will decide signed. (Note that this does not assume an honest T.)
- **(R3)** *Verifiability of valid contracts.* If an honest P_i decides signed on input (sign, *tid*, *terms*, *contr*), and later P_i receives input $(\mathsf{show}, V_j, tid, terms, contr)$ and honest V_j receives input $(\mathsf{verify}, P_i, tid, terms, contr)$ and does not receive (wakeup, *tid*, *terms*) afterwards then V_j will decide signed.
- **(R4)** *No surprises with invalid contracts.* If T is honest, and an honest P_i received input $(\mathsf{sign}, tid, terms, contr)$ but decided failed then no honest verifier V_j receiving $(\mathsf{verify}, P_{i'}, tid, terms, contr)$, for any $P_{i'}$, will decide signed.
- **(R5)** *Termination of* sign[]. If T is honest then each honest P_i that receives sign and wakeup (for the same *tid* and *terms*) will terminate eventually.
- **(R6)** *Termination of* verify[]. Each honest V_j that receives verify and then wakeup, for the same *tid* and *terms*, and each honest P_i that receives show will terminate eventually.

[4] Different sets of signatories may use the same *tid* as here *terms* will be different.

Definition 2 (Optimistic Protocol). An MPCS with third party T is called *optimistic on agreement* if in the all-honest case, if all signatories receive input (sign, *tid*, *terms*, *contr*) and none receives (wakeup, *tid*, *terms*), the protocol terminates without T ever sending or receiving any messages. It is called *optimistic on disagreement* if in the all-honest case, if some signatories do *not* receive input (sign, *tid*, *terms*, *contr*), and none receives (wakeup, *tid*, *terms*), the protocol terminates without T ever sending or receiving any messages. It is called *optimistic* if it is optimistic on agreement and on disagreement. We will call messages "optimistic" if they would appear in the "optimistic case", i.e., in a protocol execution where the third party is not needed.

4 Scheme Requiring Presence of Unwilling Parties

The following Scheme 1 solves the multi-party contract signing problem, is optimistic on agreement, and terminates in a number of locally defined rounds R as defined in Eq. 1. It requires the presence and actions of honest parties even if they do not want to sign the contract. In Section 6 we will remove this disadvantage thereby requiring one additional round. The drastic improvement towards [7] in the number of rounds results from the rule that T does not always answer requests "immediately" but has to wait for further requests, if the scheme says that such would be sent in case the sender is honest.

Ignoring all details, and somewhat simplified, Scheme 1 works as follows:

In Round 1 each signatory that wants to sign the contract starts the protocol signing its round-1-message and broadcasts it. This message will be interpreted as an "intent to sign the contract". In each subsequent round the signatory tries to collect all signatures from the previous round, countersigns this set of n signatures, and broadcasts it.[5] The result of the R-th round becomes the real contract. The messages mentioned so far build the "optimistic part". If some signatory does not want to sign the contract, it signs a reject (rather than its round-1-message), sends it to T and stops.[6]

A signatory that becomes tired of waiting for some signatures in the context of some round sends the relevant part of information received so far to the honest T, stops sending further messages, waits for T's answer (which will contain aborted or signed) and stops all actions after receiving the answer. Each answer from T is final for all honest signatories (even if T does not handle the result as final yet).

T uses the assumptions that there are at least $n - t$ honest signatories in total, and that any set of $t+1$ signatories contains at least one honest signatory. If T receives a request or a reject from a signatory P_i in the context of some round $r < R$, T uses the fact that P_i, if honest, would not send any messages in the context of a round $r' > r$ (such messages would be used as dishonesty-proofs by T). Since, if honest P_i contacted T, all other honest signatories not having contacted T so far would eventually contact T, too, either sending reject (at

[5] The real protocol does this more efficiently, in order to keep the messages short.

[6] This protocol requires presence of parties not wanting the contract. Section 6 shows how to solve this with one additional round only.

most for $r = 1$) or because a Round r or $r + 1$ message is missing (e.g., at the latest the Round-$(r + 1)$-message from honest P_i which is never sent). So it is safe for T to wait for such a number of messages.

In case P_d is dishonest and contacts T pretending missing messages without justification, it might happen that P_d will never get an answer.

If r is the highest round number in the context of which T received messages, T will consider possibly honest only those having contacted T for Rounds $r - 1$ and r.

If T receives a request, it either answers immediately (e.g., if the result is already final) or waits for further messages as long as it is sure that such will still come. T sends responses immediately if the result is already, or just becomes, clear and final: signed as soon as T has proofs that all which were sent an aborted, if any, are dishonest anyway and T has a Round-r-message with $r > 1$, seeing all the "intentions". aborted becomes final if T has proofs that one among those which will have been sent an aborted must be honest. In all other cases, after sufficient messages, T answers the non-final aborted.

Scheme 1 (Scheme Requiring Presence of Unwilling Parties)

Protocol "sign" for honest P_i:[7]

The protocol is given by the following rules; P_i applies them, repeatedly, until it *stops*. Let $c_i := (tid_i, terms_i, contr_i)$.

The protocol will proceed in locally defined rounds. R is the number of rounds as defined in Eq. 1 using the parameters n and t given in $terms_i$. Let $r := 1$ a local round counter for P_i. It will be increased only by the protocol itself or stopped by "impatience" expressed through the local service interface – there are no timeouts within the protocol itself. Let $raised\_exception :=$ false a Boolean variable indicating whether P_i contacted T or not. Both are initialized before executing any rule. Any $\mathsf{sign}_i(c_i, r, \mathsf{prev\_rnd\_ok})$ will mean P_i's confirmation that it already received all optimistic messages for c_i of the Round $r-1$ if such exists. Any $M_{r,i}$ is the complete vector of such confirmations from all signatories for the same r and c_i, and any $\mathsf{sign}_i(M_{r-1,i}, r, \mathsf{vec\_ok}))$ means P_i's confirmation that it has successfully collected such a complete vector for the previous round $r - 1$. Let $M_{0,i} :=$ nil, for all i. From $terms_i$, P_i concludes identities and public keys of the other signatories and of T. Note that information is needed in clear only if the recipient, which has to check the signature, does not have or can deviate the original information before it was signed. This holds for recipient T only. As usual, we assume implicitly that all messages contain additional implementation relevant information about the protocol and its execution.

- **Rule S0:** If $decision_i =$ reject then:
  - P_i sends $reject_i = (c_i, i, \mathsf{sign}_i(c_i, \mathsf{reject}))$ to T.
  - P_i decides failed and *stops*.
- **Rule S1:** If $raised\_exception =$ false and $r = 1$ then:

[7] These rules are similar to those in [7]. The most significant difference lies in the protocol for T.

- P_i sends $m_{1,i} := \mathsf{sign}_i(c_i, 1, \mathsf{prev\_rnd\_ok})$ to all signatories. We call those messages *optimistic*.
- From all received messages of type $m_{1,j}$, from those signatories, P_i tries to compile full and consistent vectors
  $M_{1,i} := (\mathsf{sign}_1(c_i, 1, \mathsf{prev\_rnd\_ok}), \ldots, \mathsf{sign}_n(c_i, 1, \mathsf{prev\_rnd\_ok}))$ and
  $X_{1,i} := M_{1,i}$. (NB: here P_i has verified that the sender agrees on c_i.)
  If this succeeds P_i sets $r := 2$.

At any time P_i can stop waiting for any missing $m_{1,j}$ (i.e., the user of P_i might enter wakeup any time), in which case it sets *raised_exception* := true and sends $resolve_{1,i} := (c_i, 1, i, \mathsf{sign}_i(m_{1,i}, \mathsf{resolve}))$ to T.

– **Rule S2:** If *raised_exception* = false and $2 \le r \le R$ then:
- P_i sends $m_{r,i} := (\mathsf{sign}_i(M_{r-1,i}, r, \mathsf{vec\_ok}), \mathsf{sign}_i(c_i, r, \mathsf{prev\_rnd\_ok}))$ to all signatories. We call those messages *optimistic*.
- From all received messages of type $m_{r,j}$ it tries to compile full and consistent vectors
  $M_{r,i} = (\mathsf{sign}_1(c_i, r, \mathsf{prev\_rnd\_ok}), \ldots, \mathsf{sign}_n(c_i, r, \mathsf{prev\_rnd\_ok}))$
  $X_{r,i} := (\mathsf{sign}_1(M_{r-1,i}, r, \mathsf{vec\_ok}), \ldots, \mathsf{sign}_n(M_{r-1,i}, r, \mathsf{vec\_ok}))$
  If this succeeds and if
  $r < R$ then it sets $r := r + 1$; or if
  $r = R$ then it decides signed, sets $C_i := (c_i, M_{r,i})$, and *stops*.

At any time P_i can stop waiting for any missing $m_{r,j}$. In this case P_i sets *raised_exception* := true and sends
$resolve_{r,i} := (c_i, r, i, \mathsf{sign}_i(X_{r-1,i}, \mathsf{resolve}), X_{r-1,i}, M_{r-2,i})$ to T.

– **Rule S3:** If *raised_exception* = true then:
P_i waits for a message from T (without sending any messages itself).
This can be any of $signed_{r',j} = (resolve_{r',j}, \mathsf{sign}_T(c, r', j, \mathsf{signed}))$
or $aborted_{r',j} = (c, r', j, \mathsf{sign}_T(c, r', j, \mathsf{aborted}))$.
On receiving one, P_i decides signed in the former or failed in the latter case. In case it decides signed it sets $C_i := signed_{r',j}$. In both cases, it *stops* now.

Protocol "sign" for third party T:

We assume T receives a message $resolve_{r,i}$ or $reject_{r,i}$. Let $message_{r,i}$ be that message. Let c_i be the triple (*tid*, *terms*, *contr*$_i$) contained in clear in $message_{r,i}$. Let R be as defined in Eq. 1, according to the parameters n and t as specified in *terms*.

If this is the first time T receives a message in the context of this *tid* and *terms*, T initializes Boolean variables *Signed* := false, *IsFinalResult* := false, a String variable *TheResult* := null, an Integer variable *CurrentContextRound* := 0, and sets *ReceivedQuests* := $\emptyset$, *RecogDishonestContact* := $\emptyset$, *MaybeHonestRejected* := $\emptyset$, *MaybeHonestGotAborted* := $\emptyset$, and *MaybeHonestUnanswered* := $\emptyset$.

Signed indicates T's currently guessed result, signed (by *Signed* = true) or aborted (by *Signed* = false). *IsFinalResult* tells if *Signed* is final or still might change. (Remark: A *Signed* = true will always be final, and a *Signed* = false will not change unless T has proofs that no honest signatory got any $aborted_{r',j}$.)

If *TheResult* is the final result in case of *IsFinalResult* = true. *CurrentContextRound* gives the highest number of rounds in the context of which

T has already received a message. *RecogDishonestContact* is the set of signatories which have contacted T and were identified by T as dishonest in the meanwhile. *ReceivedQuests* collects the accepted messages. *MaybeHonestRejected* is the set of signatories which sent reject to T in Round 1 and do not belong to *RecogDishonestContact*. *MaybeHonestGotAborted* is the set of signatories which earlier got from T the response aborted but do not belong to *RecogDishonestContact*. *MaybeHonestUnanswered* is the set of signatories currently waiting for an answer from T not belonging to *RecogDishonestContact*. All those sets of signatories are handled in a way that they will never overlap.

The following sequential steps process a $message_{r,i}$, and only one such message at a time. As soon as T can stop processing $message_{r,i}$ according to one of the steps, the next message can be processed. If more than one arrive, they will be processed in any order (by performing those sequential steps for each such message). $message_{r,i}$ is considered having been sent in the context of Round r, and of Round $r = 1$ in case $message_{r,i} = reject_{r,i}$. Since the model is asynchronous, T might receive messages at any time for any r and not necessarily in increasing order.

- **Step T0a:** *(T accepts only one message directly from each P_i, and only from signatories not already identified as dishonest by T.)* If $[P_i \in MaybeHonestRejected \cup MaybeHonestGotAborted \cup MaybeHonestUnanswered \cup RecogDishonestContact] \vee [message_{r,i}$ does not look like a syntactically correct message built according to the rules for P_i (e.g., if P_i does not belong to the signatories specified in *terms*)], then $message_{r,i}$ is ignored for these *tid* and *terms*, and processing that $message_{r,i}$ is stopped.
  Otherwise,
  - $ReceivedQuests := ReceivedQuests \cup \{message_{r,i}\}$ and

  in case of $message_{r,i} = reject_{r,i}$
  - $MaybeHonestRejected := MaybeHonestRejected \cup \{P_i\}$

  in case of $message_{r,i} = resolve_{r,i}$
  - $MaybeHonestUnanswered := MaybeHonestUnanswered \cup \{P_i\}$
  - $CurrentContextRound := \max(CurrentContextRound, r)$

  and (in both cases) T performs
- **Step T0b:** *(T tries to conclude dishonesty for as many as possible P_j. The sets are changed, accordingly.)* For all $P_j \in MaybeHonestRejected \cup MaybeHonestUnanswered \cup MaybeHonestGotAborted$, and for all messages $message_{r',j}$ in *ReceivedQuests* (i.e. including the new $message_{r,i}$):
  If $[r' < CurrentContextRound - 1] \vee [$*ReceivedQuests* contains directly or indirectly a message $message_{r'',j}$, $r'' > r' + 1] \vee [$*contr* contained in $message_{r',j}$ is different from any $contr'$ already directly or indirectly signed by P_j contained within another message in *ReceivedQuests*], then
  - $RecogDishonestContact := RecogDishonestContact \cup \{P_j\}$
  - $MaybeHonestRejected := MaybeHonestRejected \setminus \{P_j\}$
  - $MaybeHonestGotAborted := MaybeHonestGotAborted \setminus \{P_j\}$
  - $MaybeHonestUnanswered := MaybeHonestUnanswered \setminus \{P_j\}$
  - If $message_{r,i} = reject_{r,i}$: stop here processing $message_{r,i}$.

If $P_i \in \mathit{RecogDishonestContact}$, as a result of Step T0b, then stop processing $\mathit{message}_{r,i}$. Otherwise $\mathit{message}_{r,i}$ must be a $\mathit{resolve}_{r,i}$, and T performs the following:

– **Step T1:** *(If the result was already final then T sends it to all signatories in MaybeHonestUnanswered.)*
If $\mathit{IsFinalResult} = \mathsf{true}$ then
  - T sends to all $P_j \in \mathit{MaybeHonestUnanswered}$ the final result $\mathit{TheResult}$
  - If $\mathit{Signed} = \mathsf{false}$:
    $\mathit{MaybeHonestGotAborted} :=$
    $\mathit{MaybeHonestGotAborted} \cup \mathit{MaybeHonestUnanswered}$
  - $\mathit{MaybeHonestUnanswered} := \emptyset$
  - T stops processing $\mathit{resolve}_{r,i}$.

  Otherwise T performs

– **Step T2a:** *(If all signatories to which T sent response* aborted *or from which T received* reject*, if any, are proven dishonest and T has proofs in ReceivedQuests that all signatories wanted the contract, T responds* signed *as final result.)*
If $[\mathit{MaybeHonestGotAborted} \cup \mathit{MaybeHonestRejected} = \emptyset] \wedge$ [*ReceivedQuests* contains direct or embedded signed messages (except reject) from all n signatories to the c_i contained in $\mathit{resolve}_{r,i}$] then
  - $\mathit{IsFinalResult} := \mathsf{true}$
  - $\mathit{TheResult} := (\mathit{resolve}_{r,i}, \mathsf{sign}_T(c, r, i, \mathsf{signed}))$
  - $\mathit{Signed} := \mathsf{true}$
  - T sends to all $P_j \in \mathit{MaybeHonestUnanswered}$ the final result $\mathit{TheResult}$
  - $\mathit{MaybeHonestUnanswered} := \emptyset$
  - T stops processing $\mathit{resolve}_{r,i}$.

  Otherwise T performs

– **Step T2b:** *(If* aborted *is a safe result due to at least one honest signatory which would have it, then make it final. The proof in [1] shows that this step can only be applied if $r < R$.)* If $\sharp(\mathit{MaybeHonestRejected} \cup \mathit{MaybeHonestGotAborted} \cup \mathit{MaybeHonestUnanswered}) \geq t + 1 - \sharp(\mathit{RecogDishonestContact})$ then
  - $\mathit{IsFinalResult} := \mathsf{true}$ (and Signed stays false)
  - $\mathit{TheResult} := (c, r, i, \mathsf{sign}_T(c, r, i, \mathsf{aborted}))$
  - T sends to all $P_j \in \mathit{MaybeHonestUnanswered}$ the final result $\mathit{TheResult}$
  - $\mathit{MaybeHonestGotAborted} :=$
    $\mathit{MaybeHonestGotAborted} \cup \mathit{MaybeHonestUnanswered}$
  - $\mathit{MaybeHonestUnanswered} := \emptyset$
  - T stops Processing $\mathit{resolve}_{r,i}$.

  Otherwise T performs

– **Step T3:** *(If T considers possible that at the same time a signatory in MaybeHonestUnanswered might be honest (e.g., P_i might be honest) and all those not having sent anything to T might be dishonest (and e.g. might never send a message to T), then T must answer the current guess* aborted *though this result might not necessarily be final for T – while it will be final for any honest recipient though.)* If $[\sharp(\mathit{MaybeHonestRejected} \cup \mathit{MaybeHonestGotAborted}$
$\cup \mathit{MaybeHonestUnanswered}) \geq n - t]$ then

- T sends to all $P_j \in \mathit{MaybeHonestUnanswered}$ the result (c, r, j, $\mathsf{sign}_T(c, r, j, \mathsf{aborted})$)
- $\mathit{MaybeHonestGotAborted} := \mathit{MaybeHonestGotAborted} \cup \mathit{MaybeHonestUnanswered}$
- $\mathit{MaybeHonestUnanswered} := \emptyset$
- T stops processing $\mathit{resolve}_{r,i}$.

Otherwise T performs

– **Step T4:** *(No condition is left allowing T to immediately respond to signatories in MaybeHonestUnanswered. This means those have to wait until more signatories will ask T. The proof in [1] shows that this will happen in case MaybeHonestUnanswered actually contains honest signatories.)*
  - T stops processing $\mathit{resolve}_{r,i}$.

Protocol "verify":

If P_i wants to show a signed contract on c to verifier V it sends C_i to V. V decides signed if it receives a messages C_i from P_i such that

– **Rule V1:** $C_i = (c, (\mathsf{sign}_1(c, R, \mathsf{prev\_rnd\_ok}), \ldots, \mathsf{sign}_n(c, R, \mathsf{prev\_rnd\_ok})))$, or
– **Rule V2:** $C_i = (\mathit{resolve}_{r,j}, \mathsf{sign}_T(c, r, j, \mathsf{signed}))$, for some $r \geq 2$, and some j.

and stops. On input wakeup, V outputs verify_failed and stops.

Theorem 1 (Security of Scheme 1). *Scheme 1 is an asynchronous MPCS with third party T for any $t < n$. It is optimistic on agreement and terminates in R rounds if T is not involved, and in $R + 2$ rounds in the worst case, with R as defined in Eq. 1.*

For the detailed proof, see [1]. To show that Requirements R1 - R6 are fulfilled, it makes use of the rules for the signatories of Scheme 1, and of Lemma 1.

Lemma 1. *Consider Protocol "sign" of Scheme 1: When T has, for the first time in the context of Round $R - 1$, responded* aborted *for a $\mathit{resolve}_{R-1,i}$, then T has sent* aborted *to, or received* reject *from, or can show dishonesty-proofs for, at least $t + 1$ signatories in total.*

Lemma 1 is shown in [1], fully using all properties and cases of R in Eq. 1, depending on n and t. It is especially needed to show that R is sufficiently high so that T cannot be fooled to answer aborted to a honest signatory having contacted T in the context of Round R, while another signatory would obtain the signed contract. The proof uses also the following Lemma 2.

Lemma 2. *Consider a protocol execution p such that T, for the first time in the context of Round $R - 1$, responded* aborted*, and T has not sent* aborted *to, or received* reject *from, or can show dishonesty-proofs for, $t + 1$ signatories yet. Then $R > 2$, and T responded* aborted *in the context of each round before Round $R-1$. In the context of Round 1, T received messages (*resolve *or* reject*) from at least $n-t$ signatories. In all contexts of any two rounds r and $r+1$, $1 < r < R-2$, T received* resolve *from at least $n - t$ signatories each.*

Lemma 2 is shown in [1], using the rules for T (Steps T0 - T4) and for the signatories (Rules S0 - S3). Here the construction of R depending on n and t, according to Eq. 1 can be understood as Lemma 2 gives a hint for one of the most malicious cases significant for determining a sufficiently high R (as done for Scheme 1). If it was possible that T answered aborted for contexts including Round $R-1$, exclusively to dishonest signatories, T could be fooled to answering aborted to a honest signatory P_h having asked for Round R, while all dishonest signatories get all optimistic messages needed for their contract but just do not send one last message to P_h and pretend, towards T, missing messages in all rounds $< R$. Thus T could not distinguish this protocol execution from a crucially different protocol execution where the recipients of T's last response, sent before getting contacted by P_h, are honest and where P_h is dishonest and thus must not get the contract. Due to Lemma 1, luckily, this case cannot happen since one of them which got aborted up to context of Round $R-1$ would be honest (and thus would not send its optimistic message in Round R.)

To demonstrate that case consider $n = 13, t = 9$. According to Eq. 1, $R = 6$. We show that 5 rounds would not be sufficient: Assume $P_1, \ldots, P_9$ dishonest act towards the honest ones like in the optimistic case but do not send one last Round-R-message to them. T receives requests in the following order: For Round 1 from $P_1, \ldots, P_4$, one after the other, now T responds aborted (T3). For Round 2, a request comes from P_5, and T responds aborted immediately as the condition in T3 still holds. For Round 3, requests come from $P_6, \ldots, P_8$, and T responds aborted. T could not answer immediately because it got dishonesty-proofs for $P_1, \ldots, P_4$, reducing the left side sets in the equation for T3. For Round 4, a request comes from P_9, and T responds aborted. With 5 rounds only, the dishonest ones may have the contract, but a honest P_h asking in Round 5 would get aborted, since T has no dishonesty-proof for P_9 – it could as well have been honest, from T's point of view, thus P_h could not be sent signed. 6 rounds, however, are sufficient to prevent this (and also other) attacks.

5 Number of Messages and Rounds

Let C_s and C_b be the costs of a single and a broadcast message, respectively.

In the optimistic case, Protocol "sign" of Scheme 1 runs in R rounds, according to Eq. 1. In each round, each signatory broadcasts one message to all other signatories, resulting in costs of RnC_b. In the worst case each signatory might have one additional message exchange with T, resulting in $R + 2$ rounds and costs of $RnC_b + 2nC_s$. If one assumes that each broadcast requires $n - 1$ single messages we end up with costs of $(Rn(n-1) + 2n)C_s = O(Rn^2)C_s$. All broadcast messages of Scheme 1 have constant length, all messages sent to or by T have length $O(n)$; thus we need $O(Rn^2)$ bits only.

According to [14] any asynchronous optimistic multi-party contract signing protocol tolerating $t = n - 1$ requires a number of rounds at least linear in n. Our work shows that this lower bound actually only holds for $t = n - 1$ but not for smaller t, as – for the first time – better numbers of rounds were found for this case (e.g., $R = 2$ for $n \geq 2t + 1, R = 3$ for $n = 2t$).

6 Scheme without Presence of Unwilling Parties

Scheme 1 in Section 4 requires all honest parties to be present even if they are not willing to sign the contract. This section shows how to easily modify Scheme 1 to overcome this disadvantage, thereby spending one more round in case $n-t>1$. For $n-t=1$, Scheme 2 is identical to that in [7], and no additional round is needed.

The absence of unwilling parties and thus of reject messages implies that T cannot rely anymore on the fact that in case a signatory P_i asking in the context of Round 1 is honest, all other honest signatories would send a reject or a resolve message, too. If honest signatories do not want the contract, they would have sent messages reject in Scheme 1 but will not send anything in Scheme 2 where messages reject do not exist anymore. Therefore T has to send a response immediately for each Round-1-message (resolve only) as long as T did not receive any requests in the context of higher rounds. As soon as the first request in the context of Round 2 or higher turns up then all parties are proven to be present since they had caused sending messages $m_{r,i}$. In this case, the same situation due to the presence of honest parties is given as in Scheme 1, thus the same rules can be applied. However, an additional round is needed for Scheme 2 for the cases $n-t>1$ since only from Round 2 on, the same situation is given as from Round 1 on in Scheme 1.

Scheme 2 (Scheme Without Presence of Unwilling Parties) This Scheme is identical with Scheme 1 except for the following slight modifications.

- Unwilling signatories do not participate at all, they can simply be absent. Rule S0 is obsolete as no action is needed in case $decision_i = \mathsf{reject}$, thus no $reject_i$ will exist.
- The new number of rounds is

$$R = \begin{cases} 3, & \text{if } n \geq 2t+1; \\ 2\lfloor \frac{t+1}{n-t} \rfloor & \\ \quad + \min(2, (t+1) \mod (n-t)) + 1 & \text{if } t+2 < n \leq 2t; \\ t+2, & \text{if } n \in \{t+1, t+2\}; \end{cases} \tag{2}$$

- For T: *MaybeHonestReject* is always empty. Actions on reject and *MaybeHonestReject* are obsolete. Step T3 gets the additional condition "$CurrentContextRound = 1 \vee \ldots$", as in this case T has to answer immediately each single request.

Theorem 2 (Security of Scheme 2). *Scheme 1 is an asynchronous MPCS with third party T for any $t < n$. It is optimistic on agreement and terminates in R rounds if T is not involved, and in $R+2$ rounds in the worst case, with R as defined in Eq. 2. It does not require presence of parties not willing to sign the contract.*

The proof for this modified protocol is analoguous to the one of Scheme 1 and therefore ommitted. It makes use of the following Lemmas, replacing Lemma 1 and Lemma 2. (The case $n - t = 1$ was anyway proven in [7].)

Lemma 3. *Consider Protocol "sign" of Scheme 2: When T has, for the first time in the context of Round $R-1$, responded* aborted *for a resolve$_{R-1,i}$, then T has sent* aborted *to, or can show dishonesty-proofs for, at least $t+1$ signatories in total.*

Lemma 4. *Consider a protocol execution p such that T, for the first time in the context of Round $R-1$, responded* aborted*, and T has not sent* aborted *to, or can show dishonesty-proofs for, $t+1$ signatories yet. Then $R > 3$, and T responded* aborted *in the context of each round before Round $R-1$. In the context of Rounds 1 and 2 together, T received messages* resolve *from at least $n-t$ signatories. In all contexts of any two rounds r and $r+1, 2 < r < R-2$, T received* resolve *from at least $n-t$ signatories each.*

Scheme 2 can be transformed into an abuse-free asynchronous multi-party contract signing protocol (as defined in [13]), aided by the method presented in [7]. The result is optimistic on agreement and on disagreement and needs only 2 additional rounds and $O(n^2)$ additional messages.

7 Conclusion

Asynchronous optimistic multiparty contract signing protocols known so far required $O(t)$ rounds. In this work, protocols were found reducing the number of rounds R from $O(t)$ to $O(1)$ for all $n \geq 2t+1$, and for $n < 2t+1$, R grows remarkably slowly compared with numbers of rounds in $O(t)$: If $t \approx \frac{k}{k+1}n$ then $R \approx 2k$.

To enable absence of honest parties not wanting to sign the contract, at most one additional round is needed.

Current work is on a proof that R as defined in Eq. 1 is the lower bound with presence of willing parties, while Eq. 2 is the lower bound without requiring presence, for asynchronous optimistic multiparty contract signing protocols without error probability (assuming perfectly unforgeable digital signatures).

I would like to thank *Michael Waidner* for fruitful discussions.

References

1. B. Baum-Waidner: Optimistic Asynchronous Multi-Party Contract Signing with Reduced Number of Rounds (long version including the detailed proofs); IACR Cryptology ePrint Archive 2001, May 2001, *http://eprint.iacr.org/*
2. N. Asokan, B. Baum-Waidner, M. Schunter, M. Waidner: Optimistic Synchronous Multi-Party Contract Signing; IBM Research Report RZ 3089 (#93135), Zürich, December 1998.
3. N. Asokan, M. Schunter, M. Waidner: Optimistic Protocols for Multi-Party Fair Exchange; IBM Research Report RZ 2892, Zürich, November 1996.

4. N. Asokan, M. Schunter, M. Waidner: Optimistic Protocols for Fair Exchange; 4th ACM Conf. on Computer and Communications Security, Zürich, April 1997, 6–17.
5. N. Asokan. V. Shoup, M. Waidner: Optimistic Fair Exchange of Digital Signatures; Eurocrypt '98, LNCS 1403, Springer-Verlag, Berlin 1998, 591–606.
6. B. Baum-Waidner, M. Waidner: Asynchronous Optimistic Multi-Party Contract Signing; IBM Research Report RZ3078 (#93124), Zürich, November 1998.
7. B. Baum-Waidner, M. Waidner: Round-optimal and Abuse-free Optimistic Multi-Party Contract Signing; ICALP 2000, June 00, Geneve.
8. M. Ben-Or, O. Goldreich, S. Micali, R. L. Rivest: A Fair Protocol for Signing Contracts; IEEE Transactions on Information Theory 36/1 (1990) 40–46.
9. M. Blum: Three Applications of the Oblivious Transfer; Department of Electrical Engineering and Computer Sciences, University of California at Berkley, September 18, 1981.
10. R. Cramer, V. Shoup: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack; Crypto '98, LNCS 1462, Springer-Verlag, Berlin 1998, 13–25.
11. D. Dolev, C. Dwork, M. Naor: Non-Malleable Cryptography; 23rd Symposium on Theory of Computing (STOC) 1991, ACM, New York 1991, 542–552.
12. S. Even, Y. Yacobi: Relations Among Public Key Signature Systems; Technical Report Nr. 175, Computer Science Department, Technion, Haifa, Israel, 1980.
13. J. Garay, M. Jakobsson, P. MacKenzie: Abuse-free Optimistic Contract Signing; Crypto '99, LNCS 1666, Springer-Verlag, Berlin 1999, 449–466.
14. J. Garay, P. MacKenzie: Abuse-free Multi-party Contract Signing; Intern. Symp. on Distr. Comput. (DISC '99), LNCS 1693, Springer-Verlag, Berlin 1999, 151–165.
15. S. Goldwasser, S. Micali, R. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM J. Comput. 17/2 (1988) 281–308.
16. A. Menezes, P. van Oorschot, S. Vanstone: Handbook of Applied Cryptography; CRC Press, Boca Raton 1997.
17. S. Micali: Certified E-Mail with Invisible Post Offices; 1997 RSA Conference.
18. B. Pfitzmann, M. Schunter, M. Waidner: Optimal Efficiency of Optimistic Contract Signing; ACM PODC '98, Puerto Vallarta 1998, 113–122.
19. M. Rabin: Transaction Protection by Beacons; Journal of Computer and System Sciences 27/ (1983) 256–267.

Information-Theoretic Private Information Retrieval: A Unified Construction
(Extended Abstract)

Amos Beimel[1] and Yuval Ishai[2]

[1] Ben-Gurion University, Israel. beimel@cs.bgu.ac.il.
[2] DIMACS and AT&T Labs – Research, USA. yuval@dimacs.rutgers.edu.

Abstract. A Private Information Retrieval (PIR) protocol enables a user to retrieve a data item from a database while hiding the identity of the item being retrieved. In a *t-private*, *k-server* PIR protocol the database is replicated among k servers, and the user's privacy is protected from any collusion of up to t servers. The main cost-measure of such protocols is the *communication complexity* of retrieving a single bit of data.

This work addresses the *information-theoretic* setting for PIR, in which the user's privacy should be unconditionally protected from collusions of servers. We present a unified general construction, whose abstract components can be instantiated to yield both old and new families of PIR protocols. A main ingredient in the new protocols is a generalization of a solution by Babai, Kimmel, and Lokam to a communication complexity problem in the so-called *simultaneous messages* model.

Our construction strictly improves upon previous constructions and resolves some previous anomalies. In particular, we obtain: (1) t-private k-server PIR protocols with $O(n^{1/\lfloor(2k-1)/t\rfloor})$ communication bits, where n is the database size. For $t > 1$, this is a substantial asymptotic improvement over the previous state of the art; (2) a constant-factor improvement in the communication complexity of 1-private PIR, providing the first improvement to the 2-server case since PIR protocols were introduced; (3) efficient PIR protocols with logarithmic query length. The latter protocols have applications to the construction of efficient families of *locally decodable codes* over large alphabets and to PIR protocols with reduced work by the servers.

1 Introduction

A Private Information Retrieval (PIR) protocol allows a user to retrieve a data item of its choice from a database, such that the server storing the database does not gain information on the identity of the item being retrieved. For example, an investor might want to know the value of a specific stock without revealing which stock she is interested in. The problem was introduced by Chor, Goldreich, Kushilevitz, and Sudan [11], and has since then attracted a considerable amount of attention. In formalizing the problem, it is convenient to model the database

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 912–926, 2001.

by an n-bit string x, where the user, holding some *retrieval index* i, wishes to learn the i-th data bit x_i.

A trivial solution to the PIR problem is to send the entire database x the user. However, while being perfectly private, the *communication complexity* of this solution may be prohibitively large. Note that if the privacy constraint is lifted, the (non-private) retrieval problem can be solved with only $\lceil \log_2 n \rceil + 1$ bits of communication. Thus, the most significant goal of PIR-related research has been to minimize the communication overhead imposed by the privacy constraint. Unfortunately, if the server is not allowed to gain *any* information about the identity of the retrieved bit, then the linear communication complexity of the trivial solution is optimal [11]. To overcome this problem, Chor et al. [11] suggested that the user accesses k replicated copies of the database kept on different servers, requiring that each *individual* server gets absolutely no information on i. PIR in this setting is referred to as *information-theoretic* PIR.[1] This naturally generalizes to t-*private* PIR, which keeps i private from any collusion of t servers.

The best 1-private PIR protocols known to date are summarized below: (1) a 2-server protocol with communication complexity of $O(n^{1/3})$ bits [11]; (2) a k-server protocol with $O(n^{1/(2k-1)})$ communication bits, for any constant k (Ambainis [1] improving on [11], see also Ishai and Kushilevitz [15]); and (3) an $O(\log n)$-server protocol with $O(\log^2 n \log\log n)$ communication bits ([11], and implicitly in Beaver and Feigenbaum [4]). For the more general case of t-private PIR, the best previous bounds were obtained in [15], improving on [11]. To present these bounds, it is convenient to use the following alternative formulation of the question:

> Given positive integers d and t, what is the least number of servers for which there exists a t-private PIR protocol with communication complexity $O(n^{1/d})$?

In [15] it was shown that $k = \min\left(\lfloor dt - (d+t-3)/2 \rfloor\ ,\ dt - t + 1 - (d \bmod 2)\right)$ servers are sufficient. If t is fixed and d grows, the number of servers in this bound is roughly $(t - \frac{1}{2})d$.

No strong general lower bounds on PIR are known. Mann [20] obtained a constant-factor improvement over the trivial $\log_2 n$ bound, for any constant k. In the 2-server case, much stronger lower bounds can be shown under the restriction that the user reconstructs x_i by computing the exclusive-or of a *constant* number of bits sent by the servers, whose identity may depend on i (Karloff and Schulman [17]). These results still leave an exponential gap between known upper bounds and lower bounds. For a list of other PIR-related works the reader can consult, e.g., [7].

A different approach for reducing the communication complexity of PIR is to settle for *computational* privacy, i.e., privacy against computationally bounded servers. Following a 2-server solution by Chor and Gilboa [10], Kushilevitz and Ostrovsky [19] showed that in this setting a *single* server suffices for obtaining

[1] The term "information-theoretic PIR" may also refer to protocols which leak a negligible amount of information on i. However, there is no evidence that such a relaxation is useful.

sublinear communication, assuming a standard number-theoretic intractability assumption. The most communication efficient single-server PIR protocol to date is due to Cachin, Micali, and Stadler [9]; its security is based on a new number-theoretic intractability assumption, and its communication complexity is polynomial in $\log n$ and the security parameter. From a practical point of view, single-server protocols have obvious advantages over multi-server protocols.[2] However, they have some *inherent* limitations which can only be avoided in a multi-server setting. For instance, it is impossible for a (sublinear-communication) single-server PIR protocol to have very short queries (say, $O(\log n)$-bits long) sent from the user to the server, or very short answers (say, one bit long) sent in return. These two extreme types of PIR protocols, which can be realized in the information-theoretic setting, have found different applications (Di-Crescenzo, Ishai, and Ostrovsky [12], Beimel, Ishai, and Malkin [7]) and therefore serve as an additional motivation for information-theoretic PIR. A different, coding-related, motivation is discussed below.

Our results. We present a unified general framework for the construction of PIR protocols, whose abstract components can be instantiated to meet or beat all previously known upper bounds. In particular we obtain:

- t-private k-server PIR protocols with communication complexity $O(n^{1/\lfloor(2k-1)/t\rfloor})$. In other words, $k > dt/2$ is sufficient for the existence of a t-private k-server PIR protocol with $O(n^{1/d})$ communication. For $t > 1$, this is a substantial asymptotic improvement over the previous state of the art [15]. For example, for $t = 2$ the communication complexity of our protocol is $O(n^{1/(k-1)})$, while the communication complexity of the best previous protocol [15] is $O(n^{1/\lfloor 2k/3\rfloor})$. Our bound is essentially the best one could hope for without asymptotically improving the bounds for the 1-private case.
- A constant-factor improvement in the communication complexity compared to the 2-server protocol of [11] and its 1-private k-server generalizations from [1, 15]. In the 2-server case, this provides the first improvement since the problem was introduced in [11].
- Efficient PIR protocols with logarithmic query length: We construct a t-private k-server PIR protocol with $O(\log n)$ query bits and $O(n^{t/k+\epsilon})$ answer bits, for every constant $\epsilon > 0$. The 1-private protocols from this family were used in [7] to save computation in PIR via preprocessing, and have interesting applications, discussed below, to the construction of efficient *locally decodable codes* over large alphabets.

It is interesting to note that in contrast to previous PIR protocols, in which the user can recover x_i by reading only a *constant* number of answer bits (whose location depends only on i), most instances of our construction require the user to read *all* answer bits and remember either the queries or the randomness used to generate them. It is open whether the previous constructions of [15] (in

[2] However, for practical sizes of databases and security parameter, known multi-server (and even 2-server) protocols are much more efficient in computation and are typically even more communication-efficient than single-server protocols.

particular, the t-private protocols for $t > 1$) can be improved if one insists on the above "easy reconstruction" feature, which allows the user's algorithm to be implemented using logarithmic space.

Locally decodable codes. Information-theoretic PIR protocols have found a different flavor of application, to the construction of *locally decodable codes*. A locally decodable code allows to encode a database x into a string y over an alphabet Σ, such that even if a large fraction of y is adversarially corrupted, each bit of x can still be decoded *with high probability* by probing *few* (randomly selected) locations in y. More formally, a code $C : \{0,1\}^n \to \Sigma^m$ is said to be (k, δ, ρ)-locally decodable, if every bit x_i of x can be decoded from $y = C(x)$ with success probability $1/2 + \rho$ by probing k entries of y, even if up to a δ-fraction of the m entries of y are corrupted. Katz and Trevisan [18] have shown an intimate relation between such codes and information-theoretic PIR. In particular, any information-theoretic PIR protocol can be converted into a locally decodable code with related efficiency by concatenating the answers of all servers on all possible queries. This motivates the construction of PIR protocols with short queries.

The short-query instantiations of our PIR construction have an interesting interpretation in terms of locally decodable codes. The main focus in the works [18, 14] has been on the following question. Suppose that ρ, δ are restricted to be greater than some positive constant. Given a constant number of queries k and a *constant-size* (say, binary) alphabet Σ, what is the minimal asymptotic growth of the code length? Generalizing a PIR lower bound of [20], it is proved in [18] that for any constant k the code length must be super-linear. For the case of a linear code with $k = 2$ (non-adaptive) queries, an exponential lower bound on $m(n)$ has been obtained by Goldreich and Trevisan [14]. While no super-polynomial lower bounds are known for the case $k > 2$, the best known upper bound (obtained from PIR protocols with a single answer bit per server, see Section 6) is $m(n) = 2^{O(n^{1/(k-1)})}$, which is exponential in n. Our construction answers the following dual question: Suppose that we insist on the code being *efficient*, namely of polynomial length. Then, how small can the alphabet Σ be? More precisely, given a constant k, how small can $\Sigma_k(n)$ be such that the code length $m(n)$ is polynomial and, as before, $\rho(n), \delta(n)$ are kept constant? The short-query variants of our construction imply the following upper bound: for any constants $k \geq 2$ and $\epsilon > 0$ it suffices to let $\Sigma_k(n) = \{0,1\}^{\beta(n)}$, where $\beta(n) = O(n^{1/k+\epsilon})$.

Organization. In Section 2 we give an overview of our unified approach for constructing PIR protocols. In Section 3 we provide some necessary definitions. In Section 4 we describe a meta-construction of PIR protocols, in Section 5 we instantiate one of its crucial ingredients, and in Section 6 we derive new and old families of PIR protocols as instances of the meta-construction from Section 4. For lack of space we omitted most of the proofs. These can be found in the full version of this paper [6].

2 Overview of Techniques

At the heart of our constructions is a combination of two techniques.[3]

Reduction to polynomial evaluation. A first technique is a reduction of the retrieval problem to the problem of multivariate polynomial evaluation. Specifically, the retrieval of x_i, where the servers hold x and the user holds i, is reduced to an evaluation of a multivariate polynomial p_x, held by the servers, on a point $E(i)$, which the user determines based on i. We refer to $E(i)$ as the *encoding* of i. As observed in [5] and, more generally, in [11], the degree of p_x can be decreased by increasing the length of the encoding $E(i)$ (i.e., the number of variables in p_x). Originating in [4], different variants of this reduction have been implicitly or explicitly used in virtually every PIR-related construction. Interestingly, encodings realizing the optimal length-degree tradeoff, which were utilized in [11,12] to obtain special families of PIR protocols with short answer length, could not be used in protocols optimizing the *total* communication complexity [11,1,15]. We remedy this situation in the current work, and consequently get a constant-factor improvement to the communication complexity even in the 2-server case.

Simultaneous messages protocols for polynomial evaluation. A main ingredient in our new protocols is a generalization of a solution by Babai, Kimmel, and Lokam [3] to a communication complexity problem of computing the *generalized addressing function* in the so-called *simultaneous messages* (SM) model. Interestingly, this problem was motivated by circuit lower bounds questions, completely unrelated to privacy or coding. Towards solving their problem, they consider the following scenario. A degree-d m-variate polynomial p is known to k players, and k points $y_1, y_2, \ldots, y_k$ (each being an m-tuple of field elements) are distributed among them such that player j knows all points except y_j. An external referee knows *all* k points y_j but does not know p. How efficiently can the value $p(y_1 + y_2 + \ldots + y_k)$ be communicated to the referee if the players are restricted to simultaneously sending messages to the referee?

A naive solution to the above problem is to have one of the players send an entire description of p to the referee. Knowing all y_j, the referee can then easily compute the required output. A key observation made in [3] is that it is in fact possible to do much better. By decomposing $p(y_1 + y_2 + \ldots + y_k)$ into terms and assigning each term to a player having the least number of unknown values, it is possible to write $p(y_1 + \ldots + y_k)$ as the sum of k *lower degree* polynomials in the inputs, each known to one of the players. More precisely, player j can locally compute from its inputs a degree-$\lfloor d/k \rfloor$ polynomial p_j with its *unknown* inputs as indeterminates, such that $p(y_1 + \ldots + y_k) = p_1(y_1) + p_2(y_2) + \ldots + p_k(y_k)$. Then, by letting player j communicate the (much shorter) description of p_j, the referee can compute the required output. The amount of savings obtained by this degree reduction technique depends on the values of the parameters m,d, and k. In [3,2], due to constraints imposed by the specific problem they consider, the degree-reduction technique is applied with rather inconvenient choices of parameters. Thus, in their setting the full savings potential of the technique has

[3] A restricted use of the same approach has been made in the companion work [7].

not been realized. It turns out that in the PIR context, where there is more freedom in the choice of parameters, the full spectrum of possible tradeoffs is revealed.

It is instructive to look at three useful choices of parameters: (1) If $d = 2k-1$, then the degree of each polynomial p_j is only $\lfloor (2k-1)/k \rfloor = 1$. When $m \gg d$, this $2k-1$ savings factor in the degree makes the description size of each p_j roughly the $(2k-1)$-th root of the description size of p. (2) If $d = k-1$, the degree of each p_j becomes 0, and consequently communicating each p_j requires sending a single field element. (3) Finally, if $m \gg d$ and $d \gg k$, then the cost of communicating p_j is roughly the k-th root of that of communicating p. These three examples, respectively, turn out to imply the existence of k-server PIR protocols with: (1) both queries and answers of length $O(n^{1/(2k-1)})$; (2) queries of length $O(n^{1/(k-1)})$ and answers of length $O(1)$; (3) queries of length $O(\log n)$ and answers of length $O(n^{1/k+\epsilon})$, for an arbitrarily small constant $\epsilon > 0$.

Combining the two techniques. In the case of 1-private PIR, the two techniques can be combined in the following natural way. On input i, the user computes an encoding $y = E(i)$ and the servers compute a degree-d polynomial p_x such that $x_i = p_x(E(i))$. To generate its queries, the user "secret-shares" $E(i)$ among the servers by first breaking it into otherwise-random vectors $y_1, \ldots, y_k$ which add up to y, and then sending to each server $\mathcal{S}_j$ all vectors except y_j. Using the SM communication protocol described in the previous section, the servers communicate $x_i = p_x(y)$ to the user.

This simple combination of the two techniques is already sufficient to yield some of the improved constructions. In the remainder of this work we generalize and improve the above solution in several different ways. First, we abstract its crucial components and formulate a generic "meta-construction" in these abstract terms. Second, we instantiate the abstract components to accommodate more general scenarios, such as t-private PIR. In the full version of this paper [6], we attempt at optimizing the amount of replication in the setting of [3], i.e., use a more efficient secret-sharing scheme for distributing $E(i)$, while maintaining the quality of the solution. This motivates various extensions of the SM communication model, which may be of independent interest.

3 Definitions

Notation. By $[k]$ we denote the set $\{1, \ldots, k\}$, and by $\binom{[k]}{t}$ all subsets of $[k]$ of size t. For a k-tuple v and a set $T \subseteq [k]$, let v_T denote the restriction of v to its T-entries. By Y_j for some j we represent a variable, while by the lower letter y_j we represent an assignment to the former variable. By H we denote the binary entropy function; that is, $H(p) = -p \log p - (1-p) \log(1-p)$, where all logarithms are taken to the base 2.

Polynomials. Let $\mathrm{GF}(q)$ denote the finite field of q elements. By $F[Y_1, \ldots, Y_m]$ we denote the linear space of all polynomials in the indeterminates $Y_1, \ldots, Y_m$ over the field F, and by $F_d[Y_1, \ldots, Y_m]$ its subspace consisting of all polynomials whose *total* degree is *at most* d, and whose degree in each indeterminate is at most

$|F|-1$. (The last restriction guarantees that each polynomial in $F_d[Y_1,\ldots,Y_m]$ represents a distinct function $p : F^m \to F$.) A natural basis for this linear space consists of all *monic monomials* satisfying the above degree restrictions. The case $F = \mathrm{GF}(2)$ will be the most useful in this work. In this case, the natural basis consists of all products of at most d distinct indeterminates. Hence, $dim(F_d[Y_1,\ldots,Y_m]) = \sum_{w=0}^{d} \binom{m}{w}$ for $F = \mathrm{GF}(2)$. We denote this dimension by $\Lambda(m,d) \stackrel{\text{def}}{=} \sum_{w=0}^{d} \binom{m}{w}$. We will also be interested in $F_d[Y_1,\ldots,Y_m]$ where $|F| > d$. In this case, the dimension of the space is $\binom{m+d}{d}$.

PIR protocols. We define single-round information-theoretic PIR protocols. A k-server PIR protocol involves k servers $\mathcal{S}_1,\ldots,\mathcal{S}_k$, each holding the same n-bit string x (the database), and a user who wants to retrieve a bit x_i of the database.

Definition 1 (PIR). *A k-server PIR protocol $\mathcal{P} = (\mathcal{R},\mathcal{Q}_1,\ldots,\mathcal{Q}_k,\mathcal{A}_1,\ldots,\mathcal{A}_k,\mathcal{C})$ consists of a probability distribution $\mathcal{R}$ and three types of algorithms: query algorithms $\mathcal{Q}_j$, answering algorithms $\mathcal{A}_j$, and a reconstruction algorithm $\mathcal{C}$. At the beginning of the protocol, the user picks a random string r from $\mathcal{R}$. For $j = 1,\ldots,k$, it computes a query $q_j = \mathcal{Q}_j(i,r)$ and sends it to server $\mathcal{S}_j$. Each server responds with an answer $a_j = \mathcal{A}_j(q_j,x)$. Finally, the user computes the bit x_i by applying the reconstruction algorithm $\mathcal{C}(i,r,a_1,\ldots,a_k)$. A protocol as above is a* t-private *PIR protocol, if it satisfies: (1)* **correctness.** *The user always correctly computes x_i; (2)* **t-privacy.** *For every $i_1,i_2 \in [n]$ and $T \subseteq [k]$ such that $|T| = t$, the distributions $\mathcal{Q}_T(i_1,\mathcal{R})$ and $\mathcal{Q}_T(i_2,\mathcal{R})$ are identical.*

Linear secret-sharing. A *t-private secret-sharing scheme* allows a dealer to distribute a secret s among k players, such that any set of at most t players learns nothing on s from their joint shares, and any set of at least $t+1$ players can completely recover s from their shares. A secret-sharing scheme is said to be *linear over a field F* if $s \in F$, and the share received by each player consists of one or more linear combinations of the secret and r independently random field elements (where the same random field elements are used for generating all shares). A linear secret-sharing scheme is formally defined by a k-tuple $L = (L_1,\ldots,L_k)$ such that each L_j is a linear mapping from $F \times F^r$ to F^{ℓ_j}, where ℓ_j is the j-th player's share length. Finally, given a linear secret-sharing scheme as above, a vector in F^m will be shared by independently sharing each of its m entries. We next define two linear secret-sharing schemes that will be useful in the paper.

Definition 2 (The CNF scheme [16]). This scheme may work over any finite field (in fact, over any finite group), and proceeds as follows. To t-privately share a secret $s \in F$:

- *Additively* share s into $\binom{k}{t}$ shares, each labeled by a different set from $\binom{[k]}{t}$; that is, $s = \sum_{T \in \binom{[k]}{t}} r_T$, where the shares r_T are otherwise-random field elements.
- Distribute to each player P_j all shares r_T such that $j \notin T$.

The t-privacy of the above scheme follows from the fact that every t players miss exactly one additive share r_T (namely, the one labeled by their index set) and every set of $t+1$ players views all shares. The share vector of each party consists of $\binom{k-1}{t}$ field elements.

Definition 3 (Shamir's scheme [21]). Let $F = \mathrm{GF}(q)$, where $q > k$, and let $\omega_1, \ldots, \omega_k$ be distinct nonzero elements of F. To t-privately share a secret $s \in F$, the dealer chooses t random elements $a_1, \ldots, a_t$, which together with the secret s define a univariate polynomial $p(Y) \stackrel{\text{def}}{=} a_t Y^t + a_{t-1} Y^{t-1} + \ldots + a_1 Y + s$, and sends to the the j-th player the value $p(\omega_j)$.

4 The Meta-construction

We describe our construction in terms of its abstract general components, and specify some useful instantiations for each of these components. In Section 5 several combinations of these instantiations are used for obtaining different families of PIR protocols.

Building blocks. There are three parameters common to all of our constructions: (1) a finite *field* F, (2) a *degree* parameter d, and (3) an *encoding length* parameter m. The database x is always viewed as a vector in F^n. Some variants of our construction use an additional *block length* parameter ℓ. All variants of our construction (as well as previous PIR protocols) can be cast in terms of the following abstract building blocks:

Linear space of polynomials. Let $V \subseteq F_d[Y_1, \ldots, Y_m]$ be a linear space of degree-d m-variate polynomials such that $\dim(V) \geq n$. The three most useful special cases are:
V1: The space $F_d[Y_1, \ldots, Y_m]$ where $F = \mathrm{GF}(2)$; m and d must satisfy $\Lambda(m, d) \geq n$.
V2: The space $F_d[Y_1, \ldots, Y_m]$ where $|F| > d$; here, m and d must satisfy $\binom{m+d}{d} \geq n$.
V3: The linear subspace of $F_d[Y_1, \ldots, Y_m]$ such that $F = \mathrm{GF}(2)$ and V is spanned by the following basis of monomials. Let ℓ be an additional *block length* parameter, and let $m = \ell d$. We label the m indeterminate by $Y_{g,h}$, where $g \in [d]$ and $h \in [\ell]$. The basis of V will include all monic monomials containing exactly one indeterminate from each block, i.e., all monomials of the form $Y_{1,h_1} Y_{2,h_2} \cdots Y_{d,h_d}$. Since the number of such monomials is ℓ^d, the restriction on the parameters in this case is $\ell^d \geq n$.

Low-degree encoding. A *low-degree encoding* (with respect to the polynomial space V) is a mapping $E : [n] \to F^m$ satisfying the following requirement: There exist m-variate polynomials $p_1, p_2, \ldots, p_n \in V$ such that $\forall i, j \in [n]$, $p_i(E(j))$ is 1 if $i = j$ and is 0 otherwise. By elementary linear algebra, $\dim(V) \geq n$ is a necessary and sufficient condition for the existence of such an encoding. Given a low-degree encoding E and polynomials $p_1, p_2, \ldots, p_n$ as above, we

will associate with each database $x \in F^n$ the polynomial $p_x \in V$ defined by $p_x(Y_1, \ldots, Y_m) = \sum_{i=1}^{n} x_i p_i$. Here x is fixed, and $x_1, \ldots, x_n$ are fixed coefficients (and not variables). Note that $p_x(E(i)) = x_i$ for every $i \in [n]$ and $x \in F^n$. With each of the above linear spaces we associate a natural low-degree encoding (see [6] for proofs of validity).[4]

E1: $E(i)$ is the i-th vector in $\mathrm{GF}(2)^m$ of Hamming weight *at most* d.

E2: Let $\omega_0, \ldots, \omega_d$ be distinct field elements. Then, $E(i)$ is the i-th vector of the form $(\omega_{f_1}, \ldots, \omega_{f_m})$ such that $\sum_{j=1}^{m} f_j \leq d$.

E3: Let $(i_1, \ldots, i_d)$ be the d-digit base-ℓ representation of i (that is, $i = \sum_{j=1}^{d} i_j \ell^{j-1}$). Then, $E(i)$ is a concatenation of the length-ℓ unit vectors $e_{i_1}, e_{i_2}, \ldots, e_{i_d}$. The validity of this encoding follows by letting $p_i = Y_{1,i_1} \cdot \ldots \cdot Y_{d,i_d}$.

Linear secret-sharing scheme. Denoted by L. We will use either $L =$ **CNF** (defined in Definition 2) or $L =$ **Shamir** (defined in Definition 3). In the full version of this paper we also utilize a third scheme, which slightly optimizes **CNF**.

Simultaneous messages communication protocol (abbreviated SM protocol). The fourth and most crucial building block is a communication protocol for the following promise problem, defined by the instantiations of the previous components V, E, and L. The problem generalizes the scenario described in Section 2. The protocol, denoted P, involves a user $\mathcal{U}$ and k servers $\mathcal{S}_1, \ldots, \mathcal{S}_k$.

- *User's inputs:* Valid L-shares $y^1, \ldots, y^k$ of a point $y \in F^m$. (That is, the k vectors y^j can be obtained by applying L to each entry of y, and collecting the shares of each player.) Moreover, it may be useful to rely on the following additional promise: $y = E(i)$ for some $i \in [n]$. Most of the protocols constructed in this paper do not make use of this additional promise.
- *Servers' inputs:* All k servers hold a polynomial $p \in V$. In addition, each $\mathcal{S}_j$ holds the share vector y^j.
- *Communication pattern:* Each server $\mathcal{S}_j$ sends a single message to $\mathcal{U}$ based on its inputs p, y^j. We let β_j denote a bound on the length of the message sent by $\mathcal{S}_j$.
- *Output:* $\mathcal{U}$ should output $p(y)$.

In Section 5 we will describe our constructions of SM protocols P corresponding to some choices of the space of polynomials V, the low degree encoding E, and the linear secret-sharing scheme L.

Putting the pieces together. A 4-tuple (V, E, L, P) instantiating the above 4 primitives defines a PIR protocol $\mathrm{PIR}(V, E, L, P)$. The protocol proceeds as follows.

[4] Since the *existence* of an appropriate encoding is implied by dimension arguments, the specific encoding being employed will usually not matter. In some cases, however, the encoding can make a difference. Such a case is discussed in Section 5.

- $\mathcal{U}$ lets $y = E(i)$, and shares y according to L among the k servers. Let y^j denote the vector of shares received by $\mathcal{S}_j$.
- Each server $\mathcal{S}_j$ lets $p = p_x$, and sends a message to $\mathcal{U}$ as specified by protocol P on inputs (p, y^j).
- $\mathcal{U}$ reconstructs $x_i = p(y)$ by applying the reconstruction function specified in P to $y^1, \ldots, y^k$ and the k messages it received.

The following lemma summarizes some properties of the above protocol.

Lemma 1. $\mathrm{PIR}(V, E, L, P)$ *is a* t*-private* k*-server PIR protocol, in which the user sends* $m\ell_j$ *field elements to each server* $\mathcal{S}_j$ *and receives* β_j *bits from each server (where* ℓ_j *is the share size defined by* L *and* β_j *is the length of message sent by* $\mathcal{S}_j$ *in* P*).*

Note that the only information that a server gets is a share of the encoding $E(i)$; the t-privacy of the secret sharing scheme ensures that a collusion of t servers learns nothing on i. For the query complexity, recall that $y = E(i) \in F^m$ and the user shares each of the m coordinates of y independently. Thus, the share size of server $\mathcal{S}_j$ is $m\ell_j$, where ℓ_j is the share size defined by L for sharing one coordinate (field element).

Some perspective concerning a typical choice of parameters is in place. In the typical case where k is viewed as a constant, all ℓ_j are also constant, and so the query complexity of $\mathrm{PIR}(V, E, L, P)$ is $O(m)$. If d is constant then, for any of the three vector spaces $\mathbf{V1}, \mathbf{V2}, \mathbf{V3}$, letting $m = O(n^{1/d})$ suffices to meet the dimension requirements. Thus, when both d, k are constants, the length of the queries in $\mathrm{PIR}(V, E, L, P)$ is $O(n^{1/d})$ and the length of the answers is determined by P.

In principle, the SM component in our construction could be replaced by a more general interactive protocol. However, there is yet no evidence that such an additional interaction may be helpful. Moreover, in defining an interactive variant of the fourth primitive one would have to take special care that the privacy requirement is not violated by the interaction. In the current non-interactive framework, the privacy property is guaranteed by the mere use of a t-private secret-sharing scheme.

5 Simultaneous Messages Protocols

We next describe SM protocols corresponding to useful combinations of V, E, and L. These may be viewed as the core of the PIR protocol.

Protocol P1. Protocol **P1** will serve as our default protocol. It may be viewed as a natural generalization of the protocol from [3]. The ingredients of this protocols are the polynomial space $\mathbf{V1} = F_d[Y_1, \ldots, Y_m]$ where $F = \mathrm{GF}(2)$, the encoding **E1** which encodes i as a vector in $\mathrm{GF}(2)^m$ of weight $\leq d$, and the secret-sharing scheme **CNF**.

Lemma 2. *For* $V = \mathbf{V1}$*,* $E = \mathbf{E1}$*, and* $L = \mathbf{CNF}$*, there exists an SM protocol* **P1** *with message complexity* $\beta_j = \Lambda(m, \lfloor dt/k \rfloor)\binom{k-1}{t-1}^{\lfloor dt/k \rfloor}$.

Proof. Let $y = \sum_T y_T$ be an additive sharing of y induced by the CNF sharing, such that the input y^j of $\mathcal{S}_j$ is $(y_T)_{j \notin T}$. The servers' goal is to communicate $p(y) = p(\sum_T y_T)$ to $\mathcal{U}$. Let $Y = (Y_{T,b})_{T \in \binom{[k]}{t}, b \in [m]}$, where each variable $Y_{T,b}$ corresponds to the input bit $(y_T)_b$, whose value is known to all servers $\mathcal{S}_j$ such that $j \notin T$. Define a $\binom{k}{t}m$-variate polynomial $q(Y) \stackrel{\text{def}}{=} p(\sum_{T \in \binom{[k]}{t}} Y_{T,1}, \ldots, \sum_{T \in \binom{[k]}{t}} Y_{T,m})$. Note that q has the same degree as p, and $q((y_T)_{T \in \binom{[k]}{t}}) = p(y)$. We consider the explicit representation of q as the sum of monomials, and argue that for every monomial $Y_{T_1,b_1} Y_{T_2,b_2} \ldots Y_{T_{d'},b_{d'}}$ of degree $d' \leq d$ there exist some $j \in [k]$ such that at most $\lfloor dt/k \rfloor$ variables $Y_{T,b}$ with $j \in T$ appear in the monomial: Consider the multi-set $T_1 \cup T_2 \cup \ldots \cup T_{d'}$. This multi-set contains $d't \leq dt$ elements, thus there must be some $j \in [k]$ that appears at most $\lfloor dt/k \rfloor$ times in the multi-set. We partition the monomials of q to k polynomials $q_1, \ldots, q_k$ such that q_j contains only monomials in which the number of the variables $Y_{T,b}$ with $j \in T$ is at most $\lfloor dt/k \rfloor$. Each monomial of q is in exactly one polynomial q_j, and therefore $q(Y) = \sum_{j=1}^{k} q_j(Y)$.

We are now ready to describe the protocol **P1**. Denote by $\bar{Y}^j$ the set of variables whose values are *unknown* to the server $\mathcal{S}_j$ (that is, $\bar{Y}^j = (Y_{T,b})_{T \in \binom{[k]}{t}, j \in T, b \in [m]}$) and by $\bar{y}^j$ the corresponding values. Each $\mathcal{S}_j$ substitutes the values y^j of the variables it knows in q_j to obtain a polynomial $\hat{q}_j(\bar{Y}^j)$. The message of server $\mathcal{S}_j$ is a description of $\hat{q}_j$. The user, who knows the assignments to all variables, reconstructs the desired value by computing $\sum_{j=1}^{k} \hat{q}_j(\bar{y}^j) = q((y_T)_{T \in \binom{[k]}{t}}) = p(y)$.

MESSAGE COMPLEXITY. Recall that $\hat{q}_j$ is a degree-$\lfloor dt/k \rfloor$ multivariate polynomial with $m\binom{k}{t-1}$ variables. By the definition of q, not all monomials are possible: no monomial contains two variables $Y_{T_1,b}$ and $Y_{T_2,b}$ for some $b \in [m]$ and $T_1 \neq T_2$. Thus, to describe a possible monomial we need, for some $w \in \{0, \ldots, \lfloor dt/k \rfloor\}$, to choose w indices in $[m]$ and w sets of size t that contain j. Therefore, the number of possible monomials of $\hat{q}_j$ is at most $\sum_{w=0}^{\lfloor dt/k \rfloor} \binom{m}{w} \binom{k-1}{t-1}^w \leq \Lambda(m, \lfloor dt/k \rfloor) \binom{k-1}{t-1}^{\lfloor dt/k \rfloor}$. Since each coefficient is from GF(2), the communication is as promised. □

Protocol P2. Protocol **P2** is useful for the construction of efficient PIR protocols with short answers (see Section 6). Unlike protocol **P1**, which can be used with any combination of the parameters k, d, t, the applicability of **P2** is restricted to the case $k > dt$. That is, $k = dt + 1$ is the minimal sufficient number of servers. The first part of the following lemma is implicit in [4,5,11] and a special case of the second part is implicit in [12,13]. The proof of the lemma appears in the full version of this paper [6].

Lemma 3. *For $V = \mathbf{V2}$, $E = \mathbf{E2}$, and $L = \mathbf{Shamir}$, and assuming that $k > dt$ and $|F| > k$, there exists an SM protocol **P2** in which each server sends a single field element. Moreover, given the promise that $p(y) \in F'$ for some subfield F' of F, it suffices for each server to send a single element of F'.*

A special case of interest is when $F' = \mathrm{GF}(2)$ and F is a sufficiently large extension field of F'. In this case, each message in the SM protocol consists of a single bit.

Protocol P3. Special cases of the protocol **P3** are implicit in the 2-server PIR construction from [11] and its k-server generalization from [15]. A useful feature of this protocol is that it allows the user to compute his output by probing a small number of bits from the received messages. We only formulate this protocol for the 1-private case. Restricted generalizations to t-privacy may be obtained, using the approach of [15]. However, unlike the previous protocols, we do not know a "smooth" generalization to t-privacy. A proof of the following Lemma appears in the full version [6].

Lemma 4. *For $V =$ **V3**, $E =$ **E3**, and $L =$ **CNF**, there exists an SM protocol **P3** with message complexity $\beta_j = \ell^{\lfloor d/k \rfloor}\binom{d}{\lfloor d/k \rfloor}$ such that the user needs to read only $\binom{d}{\lfloor d/k \rfloor}$ bits from each message.*

6 Families of PIR Protocols

We now derive several explicit families of PIR protocols from the meta-construction.

Main family. Our main family of PIR protocols uses **V1**, **E1**, **CNF**, and **P1**. Protocols from this family yield our main improvements to the known upper bounds. We start with the general result, which follows from Lemmas 1 and 2, and then consider some interesting special cases.

Theorem 1. *Let m and d be positive integers such that $\Lambda(m, d) \geq n$. Then, for any k, t such that $1 \leq t < k$, there exists a t-private k-server PIR protocol with $\binom{k-1}{t}m$ query bits and $\Lambda(m, \lfloor dt/k \rfloor)\binom{k-1}{t-1}^{\lfloor dt/k \rfloor}$ answer bits per server.*

The total communication is optimized by letting $d = \lfloor (2k-1)/t \rfloor$ and $m = \Theta(n^{1/d})$. Substituting these parameters in Theorem 1 gives the following explicit bounds.

Corollary 1. *For any $1 \leq t < k$ there exist:*

- *A t-private k-server PIR protocol with $O_{k,t}(n^{1/\lfloor (2k-1)/t \rfloor})$ communication bits; this is a significant asymptotic improvement over the previous state of the art [15]. More precisely, the communication complexity is $O(\frac{k^2}{t}\binom{k}{t}n^{1/\lfloor (2k-1)/t \rfloor})$.*
- *A 1-private k-server PIR protocol with $k^2((2k-1)!n)^{1/(2k-1)} + k + k^3 = O(k^3 n^{1/(2k-1)})$ communication bits; this improves the previous best construction [15] by a constant factor which tends to e as k grows.*
- *A 1-private 2-server PIR protocol with $4(6n)^{1/3} + 2 \approx 7.27n^{1/3}$ communication bits; in comparison, the communication complexity of the best previously known 2-server protocol [11] is roughly $12n^{1/3}$.*

Another interesting case, discussed and used in [7], is when queries are short, i.e., of length $O(\log n)$; such protocols are obtained by letting $d = \theta m$, where $0 \leq \theta \leq 1/2$ is some constant. Substituting $m = (1/H(\theta) + o(1)) \log n$ and $d = \lfloor \theta m \rfloor$ in Theorem 1, and relying on the facts that $\lim_{\theta \to 0} \frac{H(\theta t/k)}{H(\theta)} = t/k$ and $\lim_{\theta \to 0} \frac{\theta}{H(\theta)} = 0$, we obtain:

Corollary 2. *For any constant integers $1 \leq t < k$ and constant $\epsilon > 0$, there exists a t-private k-server protocol with $O(\log n)$ query bits and $O(n^{t/k+\epsilon})$ answer bits. More precisely, for any $0 < \theta \leq 1/2$ one can obtain query length $\binom{k-1}{t}(1/H(\theta) + o(1)) \log n$ and answer length $n^{(H(\theta t/k)+\theta \frac{t}{k} \log \binom{k-1}{t-1})/H(\theta)+o(1)}$.*

As observed in [18], a 1-private k-server PIR protocol with query length α and answer length β gives rise to a locally decodable code of length $k \cdot 2^{\alpha}$ over the alphabet $\Sigma = \{0,1\}^{\beta}$: A string $x \in \{0,1\}^n$ is encoded by concatenating the answers of all servers on all possible queries, where x is viewed as the database. If $\alpha = O(\log n)$, then the code length is polynomial. By substituting $t = 1$ in Corollary 2 and applying the above transformation we get:

Corollary 3. *For any constant integer $k \geq 2$ and constant $\epsilon > 0$, there exist positive constants δ_k, ρ_k, such that the following holds: There is a family $C(n)$ of* polynomial-length *(k, δ_k, ρ_k)-locally decodable codes over $\Sigma(n) = \{0,1\}^{\beta(n)}$, where $\beta(n) = O(n^{1/k+\epsilon})$.*

Boolean family. We now derive the construction of the most efficient known PIR protocols with a single answer bit per server. These are obtained by using **V2**, **E2**, **Shamir**, and **P2**. Protocols from this family, utilized in [12,13], optimize similar protocols from [4,5,11] in which each answer consists of a single element from a moderately sized field. While the asymptotic communication complexity of protocols from this family is worse than that of the best unrestricted protocols, these protocols have found various applications. In particular they imply: (1) the most efficient constructions of binary locally decodable codes known to date; (2) very efficient PIR protocols for retrieving large records or "streams" of data; (3) PIR protocols with an optimal amount of total *on-line* communication (see [12]); (4) PIR protocols with poly-logarithmic amount of *on-line* work by the servers (see [7]).

Theorem 2 (Implicit in [12]). *Let m and d be positive integers such that $\binom{m+d}{d} \geq n$. Then, for any $t \geq 1$, there exists a t-private k-server PIR protocol with $k = dt+1$ servers, $\lceil \log(k+1) \rceil m$ query bits per server, and a* single *answer bit per server.*

Corollary 4. *For any constant $d, t \geq 1$ there is a t-private PIR protocol with $k = dt + 1$ servers, $O(n^{1/d})$ query bits, and a single answer bit per server.*

Cube family. Our last family of protocols generalizes the 2-server protocol from [11] and its k-server generalization from [15]. It relies on **V3**, **E3**, **CNF**, and **P3** as building blocks. The communication in these protocols is not optimal, but they have the advantage of requiring the user to read fewer bits from the answers. These protocols have the interpretation of utilizing the "combinatorial cubes" geometry which was first used in [11]. We start with the general result, and then consider interesting special cases.

Theorem 3 (Generalizing [11,15]). *Let d and ℓ be positive integers such that $\ell^d \geq n$. Then, for any $k \geq 2$ there exists a 1-private k-server PIR protocol with $(k-1)d\ell$ query bits per server and $\ell^{\lfloor d/k \rfloor}\binom{d}{\lfloor d/k \rfloor}$ answer bits per server, in which the user needs to read only $\binom{d}{\lfloor d/k \rfloor}$ bits from each answer.*

Corollary 5, which already appears in [11,15], minimizes the total communication.

Corollary 5 ([11,15]). *For any $k \geq 2$ there exists a 1-private k-server PIR protocol with $O(k^3 \cdot n^{1/(2k-1)})$ communication bits in which the user reads only $2k-1$ bits from each answer.*

As a special case, utilized in [7], we may get protocols with logarithmic query length.

Corollary 6. *For any integer $k \geq 2$ and $\delta < 1$, there exists a 1-private k-server PIR protocol with query length $O(k2^{1/\delta}\delta \log n)$ and answer length $O(n^{1/k+H(1/k)\delta})$ in which the user reads only $O(n^{H(1/k)\delta})$ bits from each answer.*

Acknowledgments. We thank Eyal Kushilevitz, Tal Malkin, Mike Saks, Yoav Stahl, and Xiaodong Sun for helpful related discussions.

References

1. A. Ambainis. Upper bound on the communication complexity of private information retrieval. In *Proc. of the 24th ICALP*, vol. 1256 of *LNCS*, pp. 401–407. 1997.
2. A. Ambainis and S. Lokam. Improved upper bounds on the simultaneous messages complexity of the generalized addressing function. In *LATIN 2000*, vol. 1776 of *LNCS*, pp. 207 – 216. 2000.
3. L. Babai, P. G. Kimmel, and S. V. Lokam. Simultaneous messages vs. communication. In *12th STOC*, vol. 900 of *LNCS*, pp. 361–372. 1995.
4. D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *STACS '90*, vol. 415 of *LNCS*, pp. 37–48. 1990.
5. D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Locally random reductions: Improvements and applications. *J. of Cryptology*, 10(1):17–36. 1997.
6. A. Beimel and Y. Ishai. Information-Theoretic Private Information Retrieval: A Unified Construction. TR01-15, Electronic Colloquium on Computational Complexity. 2001.
7. A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. In *CRYPTO 2000*, vol. 1880 of *LNCS*, pp. 56–74. 2000.
8. J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *CRYPTO '88*, vol. 403 of *LNCS*, pp. 27–35. 1990.
9. C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT '99*, vol. 1592 of *LNCS*, 402–414. 1999.
10. B. Chor and N. Gilboa. Computationally private information retrieval. In *Proc. of the 29th STOC*, pp. 304–313. 1997.

11. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *J. of the ACM*, 45:965–981. 1998.
12. G. Di-Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for private information retrieval. *J. of Cryptology*, 14(1):37–74. 2001.
13. Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *JCSS*, 60(3):592–629. 2000.
14. O. Goldreich and L. Trevisan. On the length of linear error-correcting codes having a 2-query local decoding procedure. Manuscript, 2000.
15. Y. Ishai and E. Kushilevitz. Improved upper bounds on information theoretic private information retrieval. In *Proc. of the 31th STOC*, pp. 79 – 88. 1999.
16. M. Ito, A. Saito, and T. Nishizeki. Secret sharing schemes realizing general access structure. In *Proc. of Globecom 87*, pp. 99–102. 1987.
17. H. Karloff and L. Schulman. Manuscript, 2000.
18. J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proc. of the 32th STOC*, pp. 80–86. 2000.
19. E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proc. of the 38th FOCS*, pp. 364–373. 1997.
20. E. Mann. Private access to distributed information. Master's thesis, Technion – Israel Institute of Technology, Haifa, 1998.
21. A. Shamir. How to share a secret. *CACM*, 22:612–613. 1979.

Secure Multiparty Computation of Approximations
(Extended Abstract)

Joan Feigenbaum[1⋆], Yuval Ishai[2,3], Tal Malkin[3], Kobbi Nissim[4⋆], Martin J. Strauss[3], and Rebecca N. Wright[3]

[1] Yale University, Computer Science Department, New Haven, CT 06520-8285 USA.
[2] DIMACS Center, Piscataway, NJ, USA
[3] AT&T Labs—Research, 180 Park Avenue, Florham Park, NJ 07932 USA.
[4] Weizmann Institute, Dept. of Computer Science and Applied Math. Rehovot, Israel.

Abstract. Approximation algorithms can sometimes provide efficient solutions when no efficient exact computation is known. In particular, approximations are often useful in a distributed setting where the inputs are held by different parties and are extremely large. Furthermore, for some applications, the parties want to cooperate to compute a function of their inputs without revealing more information than necessary.
If $\hat{f}$ is an approximation to f, secure multiparty computation of $\hat{f}$ allows the parties to compute $\hat{f}$ without revealing unnecessary information. However, secure computation of $\hat{f}$ may not be as private as secure computation of f, because the output of $\hat{f}$ may itself reveal more information than the output of f. In this paper, we present definitions of secure multiparty approximate computations that retain the privacy of a secure computation of f. We present an efficient, sublinear-communication, private approximate computation for the Hamming distance and an efficient private approximation of the permanent.

1 Introduction

There are an increasing number and variety of real-world applications that collect a massive amount of data and wish to make use of it. For example, massive data sets arise in physical sciences such as biology and astronomy, in marketing, in network operations, and in Web searches. The search for efficient and effective data mining algorithms is an important emerging area of research [12].

Unfortunately, many useful functions are expensive to compute. Even functions that are efficiently computable for moderately sized data sets are often not efficiently computable for massive data sets. For example, even quadratic algorithms cannot generally be considered practical on input consisting of a terabyte of data; such data sets are now routinely generated daily.[1] Besides the size of the

⋆ Part of this work was done while the author was at AT&T Labs—Research.
[1] For example, AT&T's phone network carries about 300 million calls per day, each of which generates a few kilobytes of billing data.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 927–938, 2001.

input, the distributed nature of data sets can be an obstacle to computation. A single logical data set is often stored in several remote pieces, and a computation on the data set may require prohibitively costly communication.

Another important concern, in addition to the efficiency of a computation, is its security. In a distributed setting, the pieces of a distributed data set may be controlled by different parties who wish to collaborate in order to compute some function of their data without fully revealing their piece of the data to the other parties. To that end, the parties may want to compute a function of their inputs securely, i.e., so that no party learns anything about the others' inputs except what is implied by her own output, perhaps even if some of the other parties behave maliciously. For example, rival Internet service providers often strike "peering agreements," in which each carries the other's Internet traffic at no cost, as long as the characteristics of the traffic carried by each peer for the other are comparable. The prospective peers each have data sets describing the characteristics of their own traffic, and they would like to verify the similarity of these data sets without revealing more than they have to. Several recent papers have considered the problem of privacy-preserving data mining [1,29], recognizing that it is often desirable to perform data mining without revealing unnecessary information about the data.

Separately, each of the above two concerns has been previously addressed. On one hand, when the cost of an exact computation of a function f is too high, the parties may use an *approximation* $\hat{f}$ to f. In some cases, the communication of only a small random sample from each part of a data set stored in remote pieces suffices for an approximation. In other cases, the communication of the result of a local computation depending on the entire local data set is sufficient. In both situations, the approximate computation typically requires less communication and less computation than an exact computation on the original data set. On the other hand, *secure multiparty computation* (initiated by [35,18,6,8]) allows a group of parties to compute a function f without revealing unnecessary information.

We address both concerns simultaneously: our goal is to construct approximation algorithms (more efficient than exact computation), which also maintain the privacy of the data. Note that the straightforward approach of simply computing an approximation $\hat{f}$ via a secure multiparty computation, will *not* work—it might be neither private nor efficient. Indeed, even when $\hat{f}$ itself is efficient, computing it using a general-purpose secure computation is typically very expensive. Moreover, a secure computation of $\hat{f}$ may still leak information about f that follows from the output of $\hat{f}$ itself. To illustrate this, consider an integer-valued function f and an approximation $\hat{f}$ to f that outputs $f(x_1, \ldots, x_n)$ with the last bit possibly flipped so that that last bit is 0 if x_1 is even and 1 if x_1 is odd. Then $\hat{f}(x_1, \ldots, x_n)$ is a good approximation but unnecessarily reveals the parity of x_1.

Our work. In this paper, we provide definitions of secure approximate multiparty computation that disallow the problems of information leakage discussed above, and we present protocols for several natural functions.

For massive data sets, *distance* functions are important because they give a measure of similarity between two data sets. For example, telephone companies may want to compute joint statistics on their calling data, ISPs may want to verify similar "peering" traffic characteristics, and Web search companies may want to compare their images of the Web. Because the exact computation of the Hamming and related distances requires $\Omega(n)$ communication, there has been much recent work on sublinear-communication distance approximations. For example, [3,15,21] present algorithms for efficiently approximating L^p distances between two massive data sets. However, these approximations also suffer the kind of information leakage described above. In this paper, our main technical contribution is an efficient $\tilde{O}(n^{1/2})$-communication Hamming distance approximation that does not leak unnecessary information about the parties' inputs.[2]

Approximation algorithms are also useful in the setting where the data involved is only moderate in size, but the function itself is computationally difficult. We also consider this case and provide a private approximation to the permanent and related #P-hard problems.

Related results. There are several very efficient algorithms for approximating the L^p or Hamming distance (cf. [28,2,15,21]). However, these results do not directly translate into communication-efficient private approximation protocols, as is discussed further in Section 3.

Since the presentation of an early unpublished version of our work [13], Halevi et al. [19] investigate private approximations of NP-hard functions. They show there exist natural NP-hard functions (such as the size of the minimum vertex cover in a graph) which do not admit non-trivial private approximation, although they admit good approximation algorithms without the privacy restriction. They also show that this phenomenon does not hold for all NP-hard functions by presenting an artificial NP-hard function that can be privately approximated.

The approach of constructing private sublinear-communication protocols was initiated in the context of private information retrieval [9] and further studied both in other specific contexts (e.g., [29]) and in more general contexts [31]. The latter work presents a generic methodology for transforming protocols in the communication complexity model into private protocols with low communication overhead. However, a straightforward application of their techniques to existing Hamming distance approximation protocols results in a protocol requiring superpolynomial work.

2 Secure Multiparty Approximations

In this section we define the notion of secure multiparty approximation. Further discussion of other candidate definitions and their drawbacks can be found in [14].

[2] In the full version of this paper [14], we show that if the model is relaxed to allow linear offline communication (before the parties know their inputs), then it is possible to obtain a more efficient solution, with polylogarithmic online communication, for both the Hamming distance and the L^2 distance.

Approximations. In general, an *approximation requirement* could be any binary relation between a deterministic real-valued function f, called the *target function*, and a possibly randomized real-valued function $\hat{f}$, called the *approximating function*, that defines which functions are considered good approximations. The standard approximation relation is the $\langle \epsilon, \delta \rangle$-approximation:

Definition 1. *We say that $\hat{f}$ is an* $\langle \epsilon, \delta \rangle$-approximation *to f if, for all inputs* $\mathbf{x}$, $\Pr[(1-\epsilon)f(\mathbf{x}) \leq \hat{f}(\mathbf{x}) \leq (1+\epsilon)f(\mathbf{x})] \geq 1-\delta$, *where the probability is over the randomness of $\hat{f}$.*

Secure Multiparty Computation. Secure multiparty computation allows two or more parties to evaluate some function of their inputs, such that no more is revealed to a party or a set of parties about other parties' inputs and outputs, except what is implied by their own inputs and outputs. When formally defining security, it is convenient to think of an *adversary* which tries to gain as much advantage as it can by corrupting at most t parties during the execution of the protocol. Security is then defined by requiring that whatever the adversary achieves in a *real-life* execution of the protocol it can efficiently *simulate* while corrupting at most t parties in an *ideal model*, in which a trusted party is being used to evaluate the function. Thus, the protocol prevents the adversary from gaining an extra advantage over what it could have gained in an ideal solution. We refer the reader to, e.g., [17,7,4,30] for formal definitions of secure computation in different settings, and to [14] for a somewhat simplified definition that suffices for our purposes. Although in this paper we mostly focus on the simpler case of *privacy*, i.e. security against a passive adversary, our definitions and some of our results also apply to the case of security against an active adversary who can arbitrarily modify the behavior of corrupted parties.

It is known that any polynomial-time computable (randomized) functionality can be privately computed in polynomial time [35,18]. This general feasibility result can be based on the existence of trapdoor permutations, which will be assumed throughout the paper.

Secure Approximations. We restrict our attention to approximating a *deterministic* function f, mapping an input $\mathbf{x} = (x_1, \ldots, x_m)$ (where x_i is the i-th party's input) to a non-negative integer y. A generalization to multi-output functions is straightforward.

We will start by defining a notion of *functional privacy* on which our definition will rely. Informally, an approximation function $\hat{f}$ is functionally private with respect to the target function, f, if the output of $\hat{f}$ reveals no more information about its input than f does:

Definition 2. *Let $f(\mathbf{x})$ be as above and $\hat{f}(\mathbf{x})$ be a possibly randomized function. Then $\hat{f}$ is* functionally t-private with respect to f *if there is an efficient randomized algorithm $\mathcal{S}$, called a* simulator, *such that for every* $\mathbf{x}$ *and* $1 \leq i_1, \ldots, i_t \leq m$, $\mathcal{S}((i_1, x_{i_1}), \ldots, (i_t, x_{i_t}), f(\mathbf{x}))$ *is identically distributed to $\hat{f}(\mathbf{x})$.*

Thus $\hat{f}$ is functionally 0-private if it has a simulator S such that $S(f(\mathbf{x}))$ and $\hat{f}(\mathbf{x})$ are identically distributed. In that case, we call $\hat{f}$ *functionally private* with

respect to f. Note that functional privacy implies functional t-privacy, for any $t \geq 0$. Our examples in this paper are all functionally private.

Our definition for secure approximation requires that the protocol securely computes some functionally-private approximation $\hat{f}$ of f. Since we defined $\hat{f}$ to be a single-output function, we will need to fix some convention for extending it to a multi-output function. Our default interpretation of a single-output function $\hat{f}$ in a multi-party setting assumes that a single value y is sampled from $\hat{f}(\mathbf{x})$ and is output by all parties. We stress that other conventions are possible, and a more general treatment would allow specifying an admissible collection of multi-output approximations. However, we prefer here simplicity over generality. The above discussion is formalized by the following definition, which may be instantiated with any notion of security (e.g., active or passive adversary, and computational, statistical, or perfect emulation).

Definition 3. *Let f be a deterministic function. The protocol π is a* t-secure $\langle \epsilon, \delta \rangle$-approximation protocol for f *if it t-securely computes a (possibly randomized) function $\hat{f}$, such that $\hat{f}$ is both functionally t-private with respect to f and an $\langle \epsilon, \delta \rangle$-approximation of f.*

Intuitively, the functional privacy of $\hat{f}$ with respect to f says that the input/output relation of the protocol does not reveal anything except what would have been revealed by learning f, while the secure computation of $\hat{f}$ ensures that nothing additional is revealed during the computation.

Approximations are useful both for settings in which the inputs are small but the target function is intractable and for settings in which the inputs are massive. In the latter setting, we will consider sublinear approximations to functions whose exact computation requires at least linear and at most polynomial resources. We seek protocols in which the total communication and the number of rounds are small. The parties should be able to compute their protocol responses quickly, using little storage space. Ideally, we desire that only one round of the protocol need involve raw input and that each party can compute her message for this round in just a single pass over her input, with little space and little time required to process each item. In theory, this allows the parties to regard their inputs as unbuffered data feeds that need not be stored at all, i.e., the inputs may be regarded as *data streams* (cf. [20]). Our protocols indeed satisfy these properties, requiring a single pass over the raw data, and sublinear storage, proportional to the communication complexity.

3 Sublinear Private Approximation for the Hamming Distance

In this section we present a two-party private protocol for the approximate Hamming distance. Such a protocol allows Alice, holding an input $a \in \{0,1\}^n$, and Bob, holding $b \in \{0,1\}^n$, to learn an ϵ-approximation of the Hamming distance between a, b with negligible failure probability δ, without learning additional

information about the other player's input except what follows from the Hamming distance. Our protocol requires roughly $O(n^{1/2})$ bits of communication and three rounds of interaction. By contrast, an exact computation of the Hamming distance requires $\Omega(n)$ communication, even if a small failure probability is allowed [33].

Notation. We let $d_h(a,b)$ denote the Hamming distance between a, b, and $w_h(x)$ denote the Hamming weight of an n-bit string x. The asymptotic notation $\tilde{O}(x)$ should be read as "$O(x \cdot n^{\gamma})$ for an arbitrarily small $\gamma > 0$".[3] When referring to an approximation we will often omit the parameter δ. In such a case δ should be understood to be either a constant (say 1/4) or negligible ($n^{-\omega(1)}$), as will be made clear from the context. Note that a small constant failure probability δ can always be decreased to a negligible one by $\tilde{O}(1)$ repetitions.

Approximations via Sketches. Before describing our private protocol it is instructive to consider the non-private variant of the problem. We first survey known efficient solutions for this problem, then we explain why a naive attempt to make these solutions private fails.

There are several known methods for approximating the Hamming distance using poly-logarithmic communication [3,28,11,26]. More specifically, the best $\langle \epsilon, \delta \rangle$-approximations require $O(\log n \log(1/\delta)/\epsilon^2)$ communication. These methods are based on a *sketching* approach:

Definition 4. *A* sketching protocol *for a 2-argument function* $f : \{0,1\}^n \times \{0,1\}^n \to \mathbb{N}$ *is defined by:*

- *A* sketching function, $S : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^m$ *mapping one input and a random string to a* sketch *consisting of a (presumably short) string.*
- *A (deterministic)* reconstruction function $G : \{0,1\}^m \times \{0,1\}^m \to \mathbb{R}$, *mapping a pair of sketches to an output.*

On inputs a, b, the protocol proceeds as follows. First, Alice and Bob locally compute a sketch $s_A = S(a,r)$ and $s_B = S(b,r)$ respectively, where r is a common *random input.*[4] *Then, the parties exchange sketches, and both locally output $g = G(s_A, s_B)$. We denote by $g(a,b)$ the* randomized function *defined as the output of the protocol on inputs a, b. A sketching protocol $\langle \epsilon, \delta \rangle$-approximates f if g $\langle \epsilon, \delta \rangle$-approximated f.*

We review an efficient sketching protocol for the Hamming distance [28,11]:

Example 1. **(A sketching protocol for the Hamming distance.)** Let the common random input define a 0/1-valued matrix R, with $O(\log n)$ rows and n columns, in which each entry of the i-th row (independently) takes the value 1 with probability $p_i = \beta^i$ for some constant β depending on ϵ. The sketching

[3] Following the single-parameter security definition of [17], a cryptographic security parameter can be replaced by $\tilde{O}(1)$.

[4] The common random input r can be replaced by a common pseudo-random input, whose succinct representation is picked by one party and communicated to the other.

function is defined by $S(x, R) = Rx$, where R and x are viewed as a matrix and a vector over GF(2), respectively. From the sketches Ra and Rb, the distance $d_h(a, b)$ can be approximated (roughly, by observing that $(Ra)_i = (Rb)_i$ with probability close to 1/2 if $d_h(a, b) \gg 1/p_i$ and with probability close to 1 if $d_h(a, b) \ll 1/p_i$.) The communication complexity of this sketching protocol is $O(\log n)$.

Achieving Privacy. Our goal is to obtain a sublinear-communication *private* approximation protocol for the Hamming distance. A natural approach is to seek a *general* way for converting an efficient sketching protocol approximating a function f into a private protocol approximating f.

Suppose that the (randomized) function g produced by the sketching protocol is functionally private with respect to f. This is indeed the case for the sketching protocol from Example 1 as well as for the other sketching protocols for the Hamming distance proposed in the literature. Then, to approximate f privately, it suffices to let the players privately compute g.

While a general-purpose private computation protocol can be used to evaluate g, its communication complexity will be (at least) linear in n, whereas we would like to obtain a sublinear-communication private protocol for g. At first glance, the following naive protocol seems to work. The players exchange a common random input r, locally compute a sketch based on individual inputs and r, then apply a general-purpose private computation protocol to evaluate $g = G(s_A, s_B)$ from the sketches s_A, s_B. Although efficient, this protocol generally fails to be private, even when g is functionally private with respect to f, since knowing the output of g *together with the random input* r *which was used to generate this output* can reveal additional information about the inputs. For instance, in the protocol of Example 1, Alice can deduce Rb from her input a, the output $R(a - b)$, and the common random input R.

We do not know whether the sketching method of Example 1 can be made private with sublinear communication, nor can we obtain a private protocol from any other efficient protocol for approximating the Hamming distance appearing in the literature. Underlying our solution is a combination of two different sketching protocols with small communication overhead (also referred to as *estimators*) for the Hamming distance. Both these estimators are functionally private and we show how to implement them privately with low communication overhead.

The first estimator, based on sampling, is efficient when the distance is high. Towards a private implementation of this estimator, we devise a special-purpose private protocol for comparing the bits in a random location, which may be of independent interest. This protocol uses a sublinear-communication $\binom{n}{1}$-OT protocol[5] [27,16,34,32,10], which can be based on any homomorphic encryption scheme. The second estimator is efficient when the distance is low, and, in fact,

[5] $\binom{n}{1}$-OT ("n choose 1 Oblivious Transfer") is a private 2-party protocol for the following function: the *sender*'s input is an n-bit string x and the *receiver*'s input is an index $i \in [n]$; the receiver outputs the bit x_i, and the sender has no output. Note that in the current context we only require security against a *passive* adversary.

produces an *exact* result in this case. In the following sections we describe each of the two private estimators, and then combine them to obtain the final protocol, which is private and efficient for any distance.

The High Distance Estimator. Suppose that $d = d_h(a, b)$ is guaranteed to be larger than some threshold $d_{\min}$ (which will be specified later). If $d_{\min}$ is large, then Alice and Bob can efficiently approximate d by randomly sampling a small number of bits in matching positions from their inputs. Specifically, Alice and Bob count the number of differences Δ in $s = O(\frac{n}{\epsilon^2 d_{\min}})$ randomly selected matching bits of their inputs and compute the estimate $g = \frac{\Delta \cdot n}{s}$. By a Chernoff-bound argument, g is an ϵ-approximation of d. Note that the mod-2 sum of bits in a pair of randomly selected matching positions is functionally private with respect to $d_h(a, b)$ and the functional privacy of the protocol's output follows.

We will show that for this particular sketching protocol, the output function g can be *privately* computed with a very small communication complexity. Our main tool is a private protocol for comparing a randomly sampled pair of bits. Formally, the protocol computes Sample-XOR, defined as $\mathsf{Sample\text{-}XOR}(a, b) = a_r \oplus b_r$, where $r \stackrel{R}{\leftarrow} [n]$. Note that a protocol for Sample-XOR should keep the choice of r private from each party. Figure 1 describes a private protocol for the function Sample-XOR that uses $\binom{n}{1}$-OT as a subprotocol. In it, for any $x \in \{0,1\}^n, r \in [n]$ and $m \in \{0,1\}$, we denote by $x \ll r$ a cyclic shift of x by r bits to the left, and by $x \oplus m$ the string whose i-th bit is $x_i \oplus m$.

Private-Sample-XOR

1. Alice picks a random mask $m_A \stackrel{R}{\leftarrow} \{0,1\}$ and a random shift amount $r_A \stackrel{R}{\leftarrow} [n]$.
   She computes the n-bit string $a' \stackrel{\text{def}}{=} (a \ll r_A) \oplus m_A$ Symmetrically, Bob picks $m_B \stackrel{R}{\leftarrow} \{0,1\}$ and $r_B \stackrel{R}{\leftarrow} [n]$, and computes $b' \stackrel{\text{def}}{=} (b \ll r_B) \oplus m_B$.
2. Alice and Bob invoke in parallel two $\binom{n}{1}$-OT protocols:
   - Alice retrieves $z_A \stackrel{\text{def}}{=} b'_{r_A}$ from Bob;
   - Bob retrieves $z_B \stackrel{\text{def}}{=} a'_{r_B}$ from Alice.
3. Alice sends $z'_A \stackrel{\text{def}}{=} z_A \oplus m_A$ to Bob. Bob sends $z'_B \stackrel{\text{def}}{=} z_B \oplus m_B$ to Alice. Both parties locally output $z'_A \oplus z'_B$.

Fig. 1. A private protocol for the function Sample-XOR

Lemma 1. Private-Sample-XOR *is a private protocol computing the randomized function* Sample-XOR.

Given approximation parameters ϵ, δ, our private sampling estimator for the high distance case will be implemented using $s = O((n/d_{\min}) \cdot \log(1/\delta)/\epsilon^2)$ parallel invocations of Protocol Private-Sample-XOR. As argued above, when $d = d_h(a, b) \geq d_{\min}$ then an $\langle \epsilon, \delta \rangle$-approximation of d can be computed from the

s outputs by multiplying their sum by n/s. Regardless of the distance between the inputs a, b, the view of each party in these invocations can be simulated from its input and $d_h(a, b)$. Summarizing, we have:

Lemma 2. (Private approximation for the high distance case.) *Let $\mathcal{OT}$ be an arbitrary $\binom{n}{1}$-OT protocol (with security against a passive adversary). Then, there exists a protocol for approximating $d_h(a, b)$ whose communication complexity is $\tilde{O}((n/d_{\min})/\epsilon^2)$ times that of $\mathcal{OT}$, and whose round complexity is that of $\mathcal{OT}$ plus 1, such that:*

- *The protocol is* private *with respect to the function $d_h(a, b)$ (i.e., the view of each party can be simulated from its input and $d_h(a, b)$);*
- *If $d = d_h(a, b) \geq d_{\min}$, the protocol outputs an ϵ-approximation of d with overwhelming probability.*

The Low Distance Estimator. We next consider the low distance case, where $d \leq d_{\max}$ for some threshold $d_{\max}$ to be later specified. We design two alternative private protocols for this case, each of which is based on a sketching protocol with the following properties:

- The induced function g is *almost determined* by d_h. That is, except with negligible probability, $g(a, b)$ takes a specific value determined by $d_h(a, b)$.
- The above value is equal to $d_h(a, b)$ if $d_h(a, b) \leq d_{\max}$ and "fail" otherwise.

For any sketching protocol satisfying the first property above, a private computation of g is straightforward: the parties exchange a common random input in the clear, then locally compute the sketches based on their inputs and the random input, and, finally, apply a (general-purpose) private computation protocol for evaluating the reconstruction function G on their sketches. For such a protocol to be communication-efficient, G must be computable by a small circuit. In [14], we describe two such protocols, one based on hashing and one on Reed-Solomon codes. Using either protocol, we obtain:

Lemma 3. (Private approximation for the low distance case.) *For any $1 \leq d_{\max} \leq n$, there exists a 3-round protocol π with $\tilde{O}(d_{\max})$ communication, such that:*

- *π is* private *with respect to the function $d_h(a, b)$;*
- *If $d = d_h(a, b) \leq d_{\max}$, π outputs the* exact *value of d with overwhelming probability;*
- *If $d = d_h(a, b) > d_{\max}$, π outputs "fail" with overwhelming probability.*

The Combined Protocol. The combined protocol runs both estimators setting $d_{\min} = d_{\max} = n^{1/2}/\epsilon$, and outputs the low distance output if it is not "fail", and the high distance estimator otherwise. Substituting the complexity parameters of the two subprotocols and a 2-round $\tilde{O}(1)$-communication $\binom{n}{1}$-OT protocol yields the following:

Theorem 1. *Assuming the existence of homomorphic encryption, the Hamming distance function can be privately ϵ-approximated with communication complexity $\tilde{O}(n^{1/2}/\epsilon)$ and three rounds of interaction.*

4 Efficient Approximations of #P-Hard Functions

We now turn our attention to #P-hard problems, where the goal is to achieve *polynomial-time* private approximations. Note that *artificial* privately approximable #P-hard problems are easily constructed. For example, consider any #P-hard problem $f(x)$ with output in the range $[0, 2^n]$. Then $g(x) = f(x) + 2^{2n}$ is computationally equivalent to $f(x)$, and can be privately approximated by simply outputting 2^{2n}. However, this approximation does not apply to an "interesting" quantity.

In this section, we give a private approximation of the *permanent*, the most well-known #P-complete problem. The permanent $\mathrm{per}(M)$ of a 0/1-valued matrix M counts the number of perfect matchings in the corresponding bipartite graph. Our result extends to other natural #P-complete problems, which reduce to the permanent in an approximation-preserving way. These include counting problems that arise naturally in physics, such as the number of tilings of certain lattices, or problems concerning the bond strength of molecules [25].

Approximating the Permanent. A string of results [22,5,23,24] provides efficient approximation algorithms for the permanent. We build on their technique to achieve a private approximation.

Theorem 2. *Let $f(M_1, M_2) = \mathrm{per}(M_1 + M_2)$, where M_1, M_2 are $n \times n$ matrices with n-bit non-negative entries. Then, for any $\epsilon \geq 1/\mathrm{poly}(n)$ there is a polynomial-time private ϵ-approximation for f.*

Proof (sketch): Let $M = M_1 + M_2$. Using generic secure computation, it suffices to show how to approximate $\mathrm{per}(M)$ in polynomial time in a functionally private way.

Assume that $\mathrm{per}(M) > 0$ (this can be efficiently verified). The technique of [24] uses a rapidly-mixing Markov chain to sample from the set of all perfect matchings on a graph from a distribution that is statistically indistinguishable from uniform. For our purposes, we note only that their techniques provide an efficient implementation of "coins" with success probabilities statistically indistinguishable from $p_1, \ldots, p_n$, where $\prod 1/p_i = \mathrm{per}(M)$, and, for all i, $p_i \geq 1/\mathrm{poly}(n)$. Since each probability can be efficiently estimated by sampling, a (non-private) approximation for the product, and thus for $\mathrm{per}(M)$, follows.

Note, however, that the product of approximations *does* potentially leak information about its factors (e.g., the standard deviation depends on the factors), and, thus, the product of approximations is not functionally private. Fortunately, we are able to avoid this leakage by manipulating success probabilities in various ways. Given coins with success probabilities q_0, q_1 and r, one can form coins with success probabilities $q_0 q_1$, $1 - q_0$, and $rq_1 + (1 - r)q_0$ (the latter by flipping r and then flipping q_r), without knowing any probability.

We proceed, roughly, as follows. First, we observe that if we could construct coins with success probabilities $p_i^{1/n}$, then we could also construct a coin with success probability $\prod p_i^{1/n} = 1/\mathrm{per}(M)^{1/n}$. Since $1 \leq \mathrm{per}(M)^{1/n} \leq \mathrm{poly}(n)$,

the success probability of the latter coin can be efficiently estimated, yielding an efficient and functionally-private approximation for the permanent. It thus remains to create coins with success probabilities $p_i^{1/n}$. We use the Taylor series for $x^{1/n}$ at $x = 1$. This series has the form $1 - \sum_{i \geq 1} c_i(1-x)^i$, where $\sum c_i \leq 1$ and each c_i is positive. For $1/\operatorname{poly}(n) \leq x \leq 1$, one can estimate $x^{1/n}$ by the first $kn^{O(1)}$ terms, denoted $T(x)$, with an error negligible in k. We conclude by observing that $T(x)$ is of the form constructible, using the above manipulations, from a coin with success probability x. ∎

Acknowledgements. We thank Dana Randall for suggesting some applications in Section 4 and Jessica Fong for helpful discussions and collaboration in early stages of this work.

References

1. R. Agrawal and S. Ramakrishnan. Privacy-preserving data mining. In ACM SIGMOD, 439–450, 2000.
2. N. Alon, P. Gibbons, Y. Matias, and M. Szegedy, Tracking join and self-join sizes in limited storage. In 18th PODS, 10–20, New York, 1999.
3. N. Alon, Y. Matias, and M. Szegedy, The Space Complexity of approximating the frequency moments. In 28th STOC, 20–29, 1996.
4. D. Beaver, Foundations of secure interactive computing, *CRYPTO '91, LNCS 576,* 1991.
5. A. Broder, How Hard Is It To Marry At Random? (On the approximation of the permanent), In 18th STOC, 1986, 50–58. Erratum in 20th STOC, p. 551.
6. M. Ben-Or, S. Goldwasser, and A. Wigderson, Completeness theorems for non-cryptographic fault-tolerant distributed computing. In 20th STOC, 1–10, 1988.
7. R. Canetti, Security and composition of multiparty cryptographic protocols, *Journal of Cryptology,* Vol. 13, No. 1, Winter 2000.
8. D. Chaum, C. Crépeau, and I. Damgård, Multiparty unconditionally secure protocols. In 20th STOC, 11–19, 1988.
9. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In 36th FOCS, 41–51, 1995. Journal version: *J. of the ACM*, 45:965–981, 1998.
10. C. Cachin, S. Micali, and M. Stadler, Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology: EUROCRYPT '99*, LNCS 1592, 402–414, 1999.
11. G. Cormode, M. Paterson, S. Sahinalp, and U. Vishkin, Communication complexity of document exchange. In 11th SODA, 197–206, 2000.
12. *DIMACS special year on massive data sets, 1997–1999,* `http://dimacs.rutgers.edu/SpecialYears/1997_1998/`.
13. J. Feigenbaum, J. Fong, M. Strauss, and R. Wright, Secure multiparty computation of approximations, presented at *DIMACS Workshop on Cryptography and Intractability*, 2000.
14. J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. Wright, Secure multiparty computation of approximations, available at `http://eprint.iacr.org/2001/024/`.
15. J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan, An approximate L^1-difference algorithm for massive data streams. In 40th FOCS, 501–511, 1999.

16. Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *J. of Computer and System Sciences*, 60(3):592–629, 2000. A preliminary version appeared in 30th STOC, 1998.
17. O. Goldreich, *Secure multi-party computation,* (working draft, Version 1.1), 1998. Available from `http://philby.ucsd.edu/cryptolib/BOOKS/oded-sc.html`.
18. O. Goldreich, S. Micali, and A. Wigderson, How to play any mental game. In 19th STOC, pp. 218–229, 1987.
19. S. Halevi, E. Kushilevitz, R. Krauthgamer, and K. Nissim, Private approximations of NP-hard functions. To appear, STOC 2001.
20. M. Rauch Henzinger, P. Raghavan, and S. Rajagopalan. *Computing on data streams*. Technical Report 1998-011, Digital Equipment Corporation Systems Research Center, 1998.
21. P. Indyk, Stable distributions, pseudorandom generators, embeddings and data stream computation. In 41st FOCS, 2000.
22. M. Jerrum, L. Valiant, and V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science* **43** (1986) 169–188.
23. M. Jerrum and A. Sinclair. Approximating the permanent. *SIAM Journal on Computing* **18** (1989), 1149–1178.
24. M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. STOC 2001, to appear.
25. R. Kenyon. Local statistics of lattice dimers. *Ann. IHP Prob. Stat.* **33** (1997), 591–618.
26. E. Kushilevitz and N. Nisan, *Communication complexity.* Cambridge University Press, 1997.
27. E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In 38th FOCS, 364–373, 1997.
28. E. Kushilevitz, R. Ostrovsky, and Y. Rabani, Efficient search for approximate nearest neighbor in high dimensional spaces. In 30th STOC, 614–623, 1998.
29. Y. Lindell and B. Pinkas, Privacy preserving data mining. In *Advances in Cryptology - CRYPTO '00*, LNCS 1880, 36–54, 2000.
30. S. Micali and P. Rogaway, "Secure computation", *CRYPTO '91, LNCS 576,* Springer-Verlag, 1991.
31. M. Naor, and K. Nissim. Communication preserving protocols for secure function evaluation. To appear, STOC 2001.
32. M. Naor and B. Pinkas, Oblivious transfer and polynomial evaluation. In 31st STOC, 245–254, 1999.
33. K. Pang and A. El-Gamal. Communication complexity of computing the Hamming distance. *SIAM J. on Computing*, 15(4):932–947, 1986.
34. J. P. Stern. A new and efficient all-or-nothing disclosure of secrets protocol. In *Advances in Cryptology - ASIACRYPT '98*, LNCS 1514, 357–371, 1998.
35. A. C. Yao, Protocols for secure computation. In 23rd FOCS, 160–164, 1982.
36. A. C. Yao. How to generate and exchange secrets. In 27th FOCS, 162–167, 1986.

Secure Games with Polynomial Expressions

Aggelos Kiayias[1] and Moti Yung[2]

[1] Graduate Center, CUNY, NY USA,
akiayias@gc.cuny.edu
[2] CertCo, NY USA
moti@cs.columbia.edu

Abstract. We present the first private information retrieval (PIR) scheme which is both, deterministically correct and has poly-logarithmic communication complexity. Our PIR protocol is symmetrically secure, and improves by a few orders of magnitude the known probabilistically correct poly-logarithmic scheme. This result is achieved as an application of our methodology which introduces a broad family of games, called Secure Games with Polynomial Expressions (SGPEs), that involve two interacting players: Alice and Bob. The objective of these games is the secure "interactive computation" of the value of a polynomial expression which is made up of polynomials and field elements that both players distributedly contribute to the game. The players wish to keep some or all the data (field elements and polynomials) they contribute to the game, secret and independently secure. We show that any SGPE can be played much more *efficiently* than by using generic methods, and so that no party reveals more than what it intends to. Besides the above mentioned PIR application, we present additional applications such as the "lists' intersection predicate" which is useful for secure conduct of e-commerce procedures, such as negotiation methods known as "settlement escrows" in the legal/ economics/ business literature.

1 Introduction

One of the most important results on the foundations of cryptography (suggested by Yao [Yao86], generalized to multi-party by Goldreich, Micali and Wigderson [GMW87], and characterized based on the Oblivious Transfer primitive by Kilian [Kil90]) is that given any polynomially computable function $f(x, y)$, it is possible for two parties, Alice (A for short) and Bob (B for short), to jointly compute $f(\alpha, \beta)$, with A contributing α and B contributing β, in such a way so that no party learns anything more than what can be deduced by the final output. The resulting protocols are relative to the size of the circuit that computes f that, even for simple functions, are considerably expensive to implement. Consequently, nowadays where distributed applications over the Internet are about to become a reality, it is worthwhile to seek special cases of useful function families that can accept more efficient protocol techniques (as advocated in [Gol97]).

In that spirit, Naor and Pinkas [NP99] introduced an efficient protocol for obliviously computing the value of a polynomial (Oblivious Polynomial Evalua-

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 939–950, 2001.

tion, OPE). In their setting, B possesses a polynomial P, A has a value α and wishes to obliviously compute $P(\alpha)$.

In this paper, we further investigate possibilities for efficient solutions of new useful problems in the general area of secure function evaluation by introducing a family of protocols called *Secure Games with Polynomial Expressions* (SGPEs). The general idea of our approach is to consider the joint computation of a polynomial expression that is made up of secret polynomials owned by the two players (as well as non-secret components). Player A selects an input for the expression, and wishes to obtain the value of the expression on this input. Depending on the contribution of A to the expression we can categorize SGPEs to those that A contributes only field elements to the expression (type 1), and to those that A contributes also polynomials (type 2). An example of a type-1 SGPE is the Secure Multivariate Polynomial Evaluation (SMPE): B holds a secret multivariate polynomial P, and A wishes to obtain a point in the graph of P of her choice. A Secure Nested Polynomial Game (SNPG for short) is an example of SGPE of type-2: A holds a constant number of c secret polynomials $Q_2, \ldots, Q_c$ and wants to compute $P_c(Q_c(\ldots(P_2(Q_2(P_1(\alpha))))\ldots))$ for an α of her choice, where the polynomials $P_1, P_2, \ldots, P_c$ are contributed by B.

The security conditions that we consider, are the following: A does not want to reveal anything about the data she contributes to the game, and B does not want to disclose his data beyond what is trivially inferred from A's output. In addition to the above (traditional) conditions, both players wish that if the secrecy of some of their private data is compromised or the search space of some part of the data is small, this has no effect on the secrecy of the remaining inputs (this property can be called *secret independence*). More generally players wish that their data are secure even if they are not uniformly distributed over all possible inputs.

We present an efficient construction for SGPEs of type-1 and an efficient transformation of a type-2 game to a type-1 game. We get a protocol of two flows of communication, one of which is employing an implementation of a single t-out-of-n Oblivious Transfer over values of the proper field, where t and n are small polynomial functions (in the size of the polynomial expression used in the game). Our security assumption is coding theoretic and is related to the Polynomial Reconstruction Problem. In fact, one of the basic contributions of Naor and Pinkas [NP99] is the consideration of this problem as a hard problem to base protocol security on.

Using SMPE, we provide a new Private Information Retrieval scheme (PIR) with polylogarithmic communication complexity. Our scheme is the first direct polylogarithmic Symmetric PIR that is deterministically correct and is at least five orders of magnitude better, in the polylogarithmic sense, compared to the previous polylogarithmic PIR of [CMS99]. Our PIR protocol assures correct execution always in contrast with the [CMS99]-scheme, which is a probabilistically correct protocol that exhibits a trade-off between error probability and communication complexity.

Using our construction for type-1 games we can solve a variety of other problems such as the "Lists' Intersection Predicate." In this problem, two agencies A and B, have two lists S_A, S_B respectively, and they want to check whether $S_A \cap S_B \neq \emptyset$. If this is the case, no agency wants to reveal any witness for this fact. This procedure enables negotiating parties to know that there is a common issue to be discussed without revealing mutual interests up front. This can be applied to solving (without any trusted party) the problem known as "settlement escrows." This procedure was originally proposed for pretrial negotiations (employing a trusted third party) in out of court legal settlements [GM95]. It allows two negotiating parties to figure out if their price ranges intersect and nothing more, in order to further continue with negotiating a deal. It can be applied to distributed decision-making in general e-commerce and business procedures (see [BN96]).Additional applications of our setting such as "Oblivious Negotiations" or "Oblivious Bargaining" will appear in the full version due to lack of space.

We note that trying to reduce our setting to OPE encounters a number of problems, mainly with respect to security, as the reduction fails to enforce secret-independence, a property that is necessary for the new applications. Ultimately secret-independence appears to require a stronger intractability assumption compared to the one needed for the security of OPE, which we formulate in this work. The OPE protocol has a two-flow structure for two layer computation (polynomial over data). We note that it is not at all obvious how to retain this protocol structure for our multi-layer setting, but, interestingly, we show it to be possible.

2 Preliminaries and Definitions

Let $\mathcal{P} := \{P_1, P_2, \ldots\}$ be a set of predicates, and $\mathcal{X} := \{x_1, x_2, \ldots\}$ a set of variables. An expression $\mathcal{E}$ is a rooted-DAG (direct acyclic graph) with all arcs directed towards the root specified as follows: each node is one of the following: P_i, $+$, $\cdot$, or a natural number. If a node is $+$ or $\cdot$ then it has two children, if a node is a number it has a single child; if a node is P_i then it has any non-zero number of children; each arc entering P_i is labeled by a non-zero natural number; The leaves of the DAG are selected from $\mathcal{X}$. The *value* of a path from a leaf to the root, is the product of all labels and number nodes that are in its course (and is set to 1, if there are no labels or number nodes). The *degree* of a variable is defined as the maximum path value taken over all paths from the variable node to the root. Let $\mathcal{E}$ be an expression, and let $P_1, \ldots, P_v$ denote its predicate nodes; if we map each predicate P_i to a polynomial with the same number of variables as the children of P_i and of the same degrees as the labels of its incoming arcs, an in-order traversal of $\mathcal{E}$ can be seen as a polynomial (interpreting each number node as exponentiation); we denote this polynomial by $\mathcal{E}(P_1, \ldots, P_v)$, and say that the polynomials $P_1, \ldots, P_v$ "fit into" $\mathcal{E}$.

If P is a predicate node we denote by $\text{label}(P, j)$ the label of the j-th incoming arc. Let $|\mathcal{E}|$ denote the size of the DAG (number of arcs). We define $\text{size}(\mathcal{E}) := |\mathcal{E}| + \sum_P \prod_j (\text{label}(P, j) + 1)$ where the sum is over all predicate nodes of $\mathcal{E}$. In order to store $\mathcal{E}(P_1, \ldots, P_v)$ we need $\text{size}(\mathcal{E})$ space. One of the reasons for

introducing expressions instead of talking simply about polynomials is space: if $\mathrm{coef}(P)$ denotes the number of coefficients of a polynomial P, then it holds that $\mathrm{coef}(\mathcal{E}(P_1, \ldots, P_v))$ can be exponentially large compared to $\mathrm{size}(\mathcal{E})$. In order to compute a value of $\mathcal{E}(P_1, \ldots, P_v)$ using the expression representation we need $\mathcal{O}(\mathrm{size}(\mathcal{E}))$ field operations. If $\mathcal{E}$ is an expression, denote by $d_1, \ldots, d_r$ the degrees of its variables. For a fixed constant c, we say that an expression is c-bound if $\mathrm{lcm}(d_1, \ldots, d_r) = \mathcal{O}([\mathrm{size}(\mathcal{E})]^c)$.

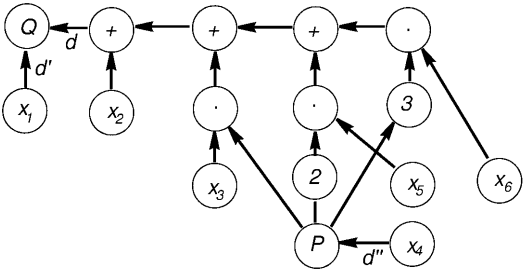


Fig. 1. Example of an expression that defines the polynomial $Q(x_1, x_2 + x_3 P(x_4) + x5(P(x_4))^2 + x_6(P(x_4))^3)$, with $\mathrm{degree}(x_2) = \mathrm{degree}(x_3) = \mathrm{degree}(x_5) = \mathrm{degree}(x_6) = d$, $\mathrm{degree}(x_1) = d'$, $\mathrm{degree}(x_4) = 3dd''$.

For the following, fix a c-bound expression $\mathcal{E}$ with v predicates and r variables. Note that we restrict the applicability of our protocol to c-bound expressions. Although we do not rule out the existence of a construction for unbounded expressions, c-bound expressions are sufficient for all applications discussed here.

A type-1 SGPE is as follows: player B has v secret polynomials $P_1, \ldots, P_v$, player A has r secret values $\alpha_1, \ldots, \alpha_r \in \mathbb{F}$ and wants to obtain $\mathcal{E}(P_1, \ldots, P_v)(\alpha_1, \ldots, \alpha_r)$. Some of the polynomials of player B may be publicly known. If $v = 1$ and $\mathcal{E}(P) := P$, then the game is called "Secure Multivariate Polynomial Evaluation" (SMPE). A type-2 SGPE is defined similarly with the only difference that some of the $P_1, \ldots, P_v$ polynomials are contributed by A. When $\mathcal{E}$ has the form $P_c(Q_c(\ldots(P_2(Q_2(P_1(x))))\ldots))$ with the P_i contributed by B, and the Q_i contributed by A then we will call this game a "Secure Nested Polynomial Game" (SNPG). Our game schema involves two flows of information, from A to B and from B to A (this latter flow employs a t-out-of-n OT). Correctness and security requirements for both types of games are as follows:

Definition 1. *Let $\mathcal{E}$ be a c-bound expression with v predicates $P_1, \ldots, P_v$, and r variables. Let $\mathcal{H}_A, \mathcal{H}_B$ denote the sequence of secrets contributed by the two players to the expression. There are two probabilistic polynomial time (PPT) algorithms $\mathcal{A}, \mathcal{B}$ and a deterministic algorithm $\mathcal{C}$ (parts of our protocol) so that: $\mathcal{C}(\mathcal{B}(\mathcal{A}(\mathcal{H}_A), \mathcal{H}_B)) = \mathcal{E}(P_1, \ldots, P_v)(\alpha_1, \ldots, \alpha_r)$ (independently of the coin tosses of $\mathcal{A}, \mathcal{B}$). Informally, $\mathcal{A}$ is used by A to hide her secrets and give them to B; B uses $\mathcal{B}$ to hide his secrets and apply them over the secrets of A; $\mathcal{C}$ is used by A*

to reconstruct the output of the protocol from the reply of B (which is obtained through a t-out-of-n OT). The computation cost is polynomial in $\text{size}(\mathcal{E})$.

Security of A. *Informally, the security of A is established by showing that B cannot deduce anything meaningful out of the protocol transcript he receives. More formally, for all PPT* $\mathcal{B}'$ *playing B's part and all probability distributions* $\mathcal{D}_A$ *for* $\mathcal{H}_A$ *there is a PPT* $\mathcal{B}''$ *such that the following is negligible:*

$$| \mathbf{Prob}[Z = \mathcal{H}_A : Z \leftarrow \mathcal{B}'(\mathcal{A}(\mathcal{H}_A))] - \mathbf{Prob}[Z = \mathcal{H}_A : Z \leftarrow \mathcal{B}''] |$$

Security of B. *Informally, security of B can be claimed by comparing with the ideal implementation. Let* $\mathcal{I}(\mathcal{H}_A, \mathcal{H}_B)$ *denote the output of player A in the ideal implementation of the protocol. Also, let* $\mathcal{T}(\mathcal{H}_A, \mathcal{H}_B)$ *be the protocol transcript obtained by player A at the end of the protocol. We show that for any PPT* $\mathcal{A}'$ *and any* $\mathcal{H}_A$ *there is a PPT* $\mathcal{A}''$ *s.t.*

$$| \mathbf{Prob}[\mathcal{A}'(\mathcal{T}(\mathcal{H}_A, \mathcal{H}_B)) = 1] - \mathbf{Prob}[\mathcal{A}''(\mathcal{I}(\mathcal{H}_A, \mathcal{H}_B)) = 1] |$$

is negligible (the probability is taken over the internal coin tosses of $\mathcal{A}', \mathcal{A}''$ *and* $\mathcal{H}_B$ *is distributed according to some probability distribution* $\mathcal{D}_B$*).*

The security of the party A for the OPE protocol of [NP99] was based on the following problem, which is also related to the security of A in our construction:

Definition 2. Polynomial Reconstruction (PR). *Given* n, k, t *and the pairs* $\{\langle z_i, y_i\rangle\}_{i=1}^{n}$ *in* $\mathbb{F}^2$, *output all* $\langle p, I\rangle$ *such that* $p \in \mathbb{F}[x]$, $\text{degree}(p) < k$, $I \subseteq \{1, \ldots, n\}$, $|I| = t$ *and* $\forall i \in I(p(z_i) = y_i)$.

PR is of prime interest in Coding Theory, since it corresponds to the decoding problem of Reed-Solomon codes. Translated in this context PR asks for all messages that agree with at least t positions of the received codeword. For a general treatment on the subject the interested reader is referred to [Ber68] or [MS77]. From the perspective of Reed-Solomon codes, we will further specialize definition 2 to require: (i) $k < n$ since k/n is the message rate of the code, and (ii) at least one solution $\langle p, I\rangle$ exists, since before the addition of the noise all pairs belong in the graph of some polynomial.

When $t \geq \frac{n+k}{2}$ then PR has only one solution and it can be found with the algorithm of Berlekamp and Welch [BW86] ($\frac{n+k}{2}$ is the error-correction bound of Reed-Solomon codes). The problem has been investigated further for smaller values of t ([Sud97,GS98,GSR95]) and it is believed that PR is hard when $t < \sqrt{kn}$ (the best algorithm known, by Guruswami and Sudan [GS98], finds all solutions when $t \geq \sqrt{kn}$). From a cryptographic perspective we are more interested in the hardness of PR on the average. It is easy to see that PR on the average (termed also noisy PR) has only one solution $\langle p, I\rangle$ such that $|I| = t$ with very high probability. It is believed that the noisy PR is not easier than the PR (see [NP99]). This is because given an instance of the PR it is possible to randomize the polynomial (but it is not known how to randomize the noise). PR was further investigated in [BN00].

Our proof of security (for player A) is based on the following problem, which we call Multisample Polynomial Reconstruction (MPR):

Definition 3. MPR. *Given n, k, t, r, and the distinct tuples $\{\langle z_i, y_{i,1}, \ldots, y_{i,r}\rangle\}_{i=1}^{n}$ so that each $\{\langle z_i, y_{i,\ell}\rangle\}_{i=1}^{n}$ is a noisy PR instance with parameters n, k, t and solution $\langle p_\ell, I\rangle$, find $\langle p_1, \ldots, p_r, I\rangle$.*

MPR appears to be not much easier than PR, a fact that is justified similarly to the case of PR vs. noisy PR: given an instance of PR it is possible to randomize the polynomial r times and get a version of MPR, but as before it is not apparent how to replicate and randomize the noise. We will formulate this as a complexity assumption:

Complexity Assumption. For any r there are n, k, t polynomially related parameters so that any probabilistic algorithm solving the MPR has negligible success probability in n.

Solving MPR either involves using techniques against a specific noisy PR instance that is included in the MPR instance (since the recovery of some $\langle p_\ell, I\rangle$ immediately implies the recovery of $\langle p_1, \ldots, p_r, I\rangle$) or in a more direct fashion trying to take advantage of the relation between the noisy PR instances included in the MPR instance. The best algorithm for solving PR is [GS98], which succeeds when $t \geq \sqrt{kn}$. Solving MPR directly has been discussed recently in [KY01] and succeeds for choices of $r > \frac{n}{t}$. As a result the current state of the art suggests that MPR is hard when $ct < \sqrt{kn}$ and $c'r < \frac{n}{t}$ for some $c, c' > 1$.

To complete this section, let us comment briefly on the relation of SGPEs and OPE. In particular if it is possible to simulate a SGPE using OPE; there are two possibilities: (1) if only univariate polynomials appear in the expression the two players can use many individual OPEs to obtain intermediate results and finally player A will compose the final output. Nevertheless to conform to the security requirements randomization of the partial results is necessary something that appears to be hard unless the expression degenerates to an affine transformation. (2) The case of multivariate polynomial evaluation e.g. $P(\alpha, \beta)$, it can be performed by OPE as follows: A sends to B random $s(x), s'(x) \in \mathbb{F}[x]$ s.t. $s(x_0) = \alpha$ and $s'(x_0) = \beta$ (x_0 is kept secret by A); A and B engage in OPE so that A obtains $P(s(x_0), s'(x_0))$. This approach has the deficiency that the values contributed by A are not "independently secure", i.e. partial knowledge of some of the values (or a small search-space for one of the values) can lead to the recovery of all secret input of A with non-negligible probability.

3 SPGEs of Type 1

In the following construction, a t-out-of n OT protocol is used as a primitive.

- *Protocol parameter*: a c-bound expression $\mathcal{E}$ of v predicates.
- *Input of B*: Polynomials $P_1, \ldots, P_v$ that fit into $\mathcal{E}$.
- *Input of A*: r elements of $\mathbb{F}$, $\alpha_1, \ldots, \alpha_r$.
- *Output of A*: $\mathcal{E}(P_1, \ldots, P_v)(\alpha_1, \ldots, \alpha_r)$.
- *Security parameters*: n, l.
- Let $P(x_1, \ldots, x_r) := \mathcal{E}(P_1, \ldots, P_v)(x_1, \ldots, x_r)$, and denote by d_ℓ the degree of x_ℓ in P. Set $d := lr\mathrm{lcm}(d_1, \ldots, d_r)$, and $k := \min_\ell \frac{d}{rd_\ell} + 1$.

Step 1. A generates r instances of the noisy PR, $\{\langle z_i, y_{i,\ell}\rangle\}_{i=1}^{n}$ with solution $\langle p_\ell, I\rangle$, such that $p_\ell(0) := \alpha_\ell$, degree$(p_\ell) = k-1$, $z_i \neq 0$ and $z_i \neq z_j$ for all i, j, $j \neq i$. Then, A, forms the $(r+1)$-tuples $\{\langle z_i, y_{i,1}, \ldots, y_{i,r}\rangle\}_{i=1}^{n}$, and she sends them to B.

Step 2. B hides P in a random polynomial Q: Let $C, C' \in \mathbb{F}[x]$ be random polynomials of degree d such that $C(x) = C'(x) = 0$. Define a polynomial $Q \in \mathbb{F}[x_0, x_1, \ldots, x_r]$ as follows: $Q(x_0, \ldots, x_r) = P(x_1, \ldots, x_r) + C(x_0) + x_1^{d_1} \ldots x_r^{d_r} C'(x_0)$. The storage space needed for Q is size$(\mathcal{E}) + 2d$. Computing a value of Q requires $\mathcal{O}(\text{size}(\mathcal{E}) + d)$ field operations. For each tuple $(z_i, y_{i,1}, \ldots, y_{i,r})$ B computes the value $Q(z_i, y_{i,1}, \ldots, y_{i,r})$. Note that the polynomial $R(x) := Q(x, p_1(x), \ldots, p_r(x))$ on 0 gives $R(0) = P(\alpha_1, \ldots, \alpha_r)$. The degree of R is $d_R = d + d_1 d_{p_1} + \ldots + d_r d_{p_r} \leq 2d$. Therefore, if A learns $t := 2d+1$ values of R, she can interpolate it and compute $R(0)$.

Step 3. A and B engage in a t-out-of n OT in which A chooses to learn the values $Q(z_i, p_1(z_i), \ldots, p_r(z_i))$. Now A knows $2d+1$ values of the polynomial R and can interpolate it to compute $R(0) = P(\alpha_1, \ldots, \alpha_r)$.

Implementation and Complexity. Clearly, A can compute $P(\alpha_1, \ldots, \alpha_r)$ for any $\alpha_1, \ldots, \alpha_r$ of her choice. The time-complexity of the protocol is $\mathcal{O}(rn + d \log^2 d + f_A(t, n))$ for player A and $\mathcal{O}(nd + n\text{size}(\mathcal{E}) + f_B(t, n))$ for player B, where $f_A(t, n), f_B(t, n)$ denotes the running time of the t-out-of n OT protocol for each player respectively. The communication complexity is $\mathcal{O}(rn + c(t, n))$ where $c(t, n)$ is the communication complexity of the t-out-of n OT. Regarding the security parameters, in section 3.1 we show that $n = \mathcal{O}(rd + d^2/l)$ is sufficient; l relates to the value k and is chosen so that k is large enough so that player B is not be able to find $p_1, \ldots, p_r$ by brute-force in $\min\{\binom{n}{t}, \binom{n}{k}\}$ steps. We point here that if the expression $\mathcal{E}$ is 0-bound, then the complexity of player A does not depend on the size of the expression. For a t-out-of-n OT protocol the reader is referred to e.g. [NP99] where t-out-of n OT is efficiently and unconditionally reduced to 1-out-of-2 OT.

3.1 Security of A

The security of A is based on the hardness of MPR as the following theorem reveals:

Theorem 1. *If* B *breaks the security of* A *in our protocol then, assuming that the underlying t-out-of-n OT is secure,* MPR *is polynomial time for parameters* $n, k-1 := \min_\ell \frac{d}{rd_\ell}, t := 2d+1, r$.

By a suitably large choice of the security parameter n we can enforce the security of A under the MPR-complexity-assumption (provided that the security parameter l is large enough to withstand a brute-force attack – see previous section). Both $ct < \sqrt{kn}$ and $c'r < \frac{n}{t}$ should be satisfied (the parameters c, c' allow for small improvements to the results against MPR). It is easy to see that it suffices to select $n = \mathcal{O}(rd + d^2/l)$.

3.2 Security of B

The security of player B is established by showing that the output of player A out of a protocol execution (the protocol transcript obtained by A) is essentially identical to what she gets in an ideal implementation. This holds true independently of A's behavior. In an ideal implementation, A gives to a trusted third party C all information send to B in step 1 of the protocol together with the randomness she used — note that this reveals her secret values $\alpha_1, \ldots, \alpha_r$. Player B gives to C its secret input $P_1, \ldots, P_v$. In turn, C returns to A, either a value of $\mathcal{E}(P_1, \ldots, P_v)(x_1, \ldots, x_r)$ or a linear combination of some values of $\mathcal{E}(P_1, \ldots, P_v)(x_1, \ldots, x_r)$ (the exact formulation is given in the full version).

Lemma 1. *There is a PPT $\mathcal{G}$ that given the output of the ideal implementation of the protocol for player A, and all information available to player A, generates a protocol transcript that is statistically indistinguishable from legitimate protocol transcripts generated during normal operation, under the assumption that t-out-of-n OT can be implemented ideally.*

Theorem 2. *Our construction is secure with respect to player B under the assumption that the underlying t-out-of-n OT is secure.*

4 SGPEs of Type 2

In this section we present a transformation of type-2 games to type-1 games. First we deal with SNPGs: we will consider only the two round case and it will become clear how to generalize to any constant number of rounds. Suppose B possesses the secret polynomials $P_2, P_1 \in \mathbb{F}[x]$ and A the secret polynomial $Q_2 \in \mathbb{F}[x]$ of degree δ (known to B). A wants to compute $P_2(Q_2(P_1(\alpha)))$ for an α of her choice. B defines the expression $\mathcal{E}(P_1, P_2)(x_0, \ldots, x_\delta, x) = P_2(x_0 + x_1 P_1(x) + \ldots x_\delta (P_1(x))^\delta)$.

If $Q_2(x) = a_0 + a_1 x + \ldots a_\delta x^\delta$ then A, using the type-1 protocol, can compute the value $\mathcal{E}(P_2, P_1)(a_0, \ldots, a_\delta, \alpha)$ for an α of her choice. Now by the definition: $\mathcal{E}(P_2, P_1)(a_0, \ldots, a_\delta, \alpha) = P_2(a_0 + a_1 P_1(\alpha) + \ldots a_\delta (P_1(\alpha))^\delta) = P_2(Q_2(P_1(\alpha)))$.

The case of any type-2 game can be sketched as follows: player A should obtain $\mathcal{E}(P_1, \ldots, P_v, Q_1, \ldots, Q_{v'})(\alpha_1, \ldots, \alpha_r)$ where the polynomials $P_1, \ldots, P_v$ are contributed by B, and the values $\alpha_1, \ldots, \alpha_r$, and polynomials $Q_1, \ldots, Q_{v'}$ are contributed by A. For simplicity we assume that the polynomials Q_i are univariate. Let the degree of Q_i be δ_i. B substitutes in the expression $\mathcal{E}$ each occurrence of $Q_i(V)$ with $x_0 + x_1 V + \ldots x_{\delta_i} V^{\delta_i}$ for all $i = 1, \ldots, v'$. The resulting expression is $\mathcal{E}'$. Note that $\mathcal{E}'$ is independent of the sequence of the substitutions (each substitution works on a disjoint portion of the DAG). It is not hard to show that $|\mathcal{E}'| = \mathcal{O}(\text{size}(\mathcal{E}))$, and consequently $\text{size}(\mathcal{E}') = \mathcal{O}(\text{size}(\mathcal{E}))$. Note also that if $\mathcal{E}$ is c-bound then $\mathcal{E}'$ is also c-bound. By engaging in type-1 game with $\mathcal{E}'$, player A can "plug-in" all the coefficients of her polynomials, along with the values $\alpha_1, \ldots, \alpha_r$, and therefore the type-2 game transforms to a type-1 game.

Theorem 3. *The correctness and security of our construction for type-1 games, implies the correctness and security of the type-2 protocol described above.*

We note that in general, SNPGs are not produced by c-bound expressions. An expression for an SNPG is c-bound only if the number of polynomials contributed by both players is constant (constant nesting).

5 Private Information Retrieval

In Private Information Retrieval (PIR for short), the database prober, wants to obtain a bit or an object of her choice from a database of size N, without revealing her choice to the database moderator. The problem was introduced in [CGKS95]. A PIR can be seen as a 1-out-of-N OT with the additional restriction that we are interested in achieving prober time complexity which is sublinear in N, and more specifically sublinear communication complexity. Note that in a PIR scheme the security of the database is not enforced; something that happens in a Symmetric PIR – SPIR for short, [GIKM98]. Communication complexity of $\mathcal{O}(N^{1/k})$ in [CGKS95] (replication of the databases), and later $\mathcal{O}(N^c)$ in [CG97] (computational setting – cPIR) was shown. In [KO97] replication was dropped as a requirement (for the computational setting), and in [CMS99] the first cPIR with polylogarithmic communication complexity was presented. For any cPIR, it seems inevitable that the communication complexity is polynomial in some security parameter l. A polylogarithmic PIR has communication complexity of $\mathcal{O}(\text{polylog}(N))$, therefore it is meaningful to require $l = \mathcal{O}(\text{polylog}(N))$ also. In [CMS99], communication is polynomial in the security parameter l and the moderator can break the security of the prober by an $\mathcal{O}(2^{cl})$ computation (provided that the underlying security assumption is true); therefore by choosing $l = \Omega(\delta^2)$, where $\delta = \log N$, breaking the security of the prober becomes super-polynomial in N. Here, we present the first direct (computational) SPIR protocol that has polylogarithmic communication complexity. We achieve substantial improvements compared to the result of [CMS99]:

1. The correctness of our PIR protocol is deterministic, rather than probabilistic as in [CMS99]. Note that in the [CMS99]-PIR reducing the error probability results in asymptotic increase of the communication complexity.
2. The communication complexity of our SPIR protocol is $\mathcal{O}(hl^2\delta^3)$, where h denotes the number of bits that are required to store a single object of the database. The choice $l = \Omega(\delta)$ is sufficient in order to ensure that the moderator needs to spend super-polynomial time in N for the search. If we set $l := \delta$ the communication complexity of our scheme is $\mathcal{O}(h\delta^5)$. The communication of the [CMS99]-PIR is $\mathcal{O}(hl^f)$ where $f \geq 5$ and depends on the underlying Φ-Hiding Assumption (its second constant). If $f = 5$, and $l = \delta^2$ then the communication of the PIR is $\mathcal{O}(h\delta^{10})$. Increasing l to achieve stronger security for the prober, yields larger asymptotic speed-up for our PIR-scheme.

The time-complexity of the two parties is low: $\mathcal{O}(l^2\delta^3)$ for the prober, and $\mathcal{O}(Nl\delta^2)$ for the moderator; our construction requires an $\mathcal{O}(N^2)$ pre-processing

stage by the database moderator that needs to be performed only once, prior to servicing any number of requests.

Note that we deal directly with a database containing words rather than bits. Let Δ be a database consisting of $N := 2^\delta$ words, $\Delta := \{w_0, \ldots, w_{N-1}\}$. Define a polynomial $P(x_1, \ldots, x_\delta) := \sum_{j_1,\ldots,j_\delta} a_{j_1 j_2 \ldots j_\delta} x_1^{j_1} x_2^{j_2} \ldots x_\delta^{j_\delta}$, where each $j_\ell \in \{0,1\}$, $\ell = 1, \ldots, \delta$. Let $v(j_1, \ldots, j_\delta) := j_\delta + 2j_{\delta-1} + \ldots + 2^{\delta-1} j_1$. We write $\langle j_1, \ldots, j_\delta \rangle \prec \langle i_1, \ldots, i_\delta \rangle$ to denote the coordinate-wise ordering of bit-strings. The coefficient $a_{j_1 j_2 \ldots j_\delta}$ of P is defined recursively as follows: $a_{j_1 j_2 \ldots j_\delta} := w_{v(j_1,j_2,\ldots,j_\delta)} - \sum_{\langle j'_1, j'_2, \ldots, j'_\delta \rangle \prec \langle j_1, j_2, \ldots, j_\delta \rangle} a_{j'_1 j'_2 \ldots j'_\delta}$ (note that $a_{00\ldots0} := w_0$).

In our PIR protocol, A plays the role of the prober and player B is the moderator of the database. B prepares P during a pre-processing stage. A wants to obtain the word w_q of the database. Let $\langle j_1, \ldots, j_\delta \rangle$ be the binary representation of q. By using the type-1 protocol for SMPE, A obtains the value $P(j_1, \ldots, j_\delta)$ which is equal to w_q.

Theorem 4. *The scheme above is a deterministically correct SPIR scheme with polylogarithmic communication complexity.*

We point out that the multivariate polynomial setting is suitable for PIR, since only in such a polynomial it is possible to directly store "exponentially" many words while at the same time keeping the degree logarithmically small w.r.t. the number of coefficients. This is what allows the complexity of the prober to be sublinear in the database size, as the prober has to spend polynomial time in the degree of the polynomial.

6 Lists' Intersection Predicate

The List Intersection Problem was introduced and solved in [NP99]: two agencies holding two lists, jointly compute their intersection, without revealing any elements not common to both lists. Here we consider a different setting for this problem where even the common part needs to remain secret. More specifically, the two agencies have a number of lists and want to *check* whether there exist any common items in these lists; if this is the case no party should get any information about these elements. This makes it possible for two parties to discover whether they are holding the same elements *without* revealing them if this is the case.

Assume that B has a collection of sets $S_B^1, \ldots, S_B^v$ and A has a collection of sets $S_A^1, \ldots, S_A^v$. A wants to compute the truth-value of the following predicate:

$$(S_A^1 \cap S_B^1 \neq \emptyset) \wedge (S_A^2 \cap S_B^2 \neq \emptyset) \wedge \ldots \wedge (S_A^v \cap S_B^v \neq \emptyset)$$

B agrees that A can learn the truth-value of the predicate however he does not want to let A find out anything more (e.g. in case $S_A^1 \cap S_B^1 \neq \emptyset$, A should not find a witness for this fact). For simplicity, we assume that $\forall j, |S_B^j| = L$. Let $S_A^1 = \{\alpha_1, \ldots, \alpha_{k_1}\}$ and for $j = 2, \ldots, v$, $S_A^j = \{\alpha_{k_{j-1}} + 1, \ldots, \alpha_{k_j}\}$. Let $k :=$

$k_v = \sum_{j=1}^{v} |S_A^j|$. B computes k polynomials p_i such that $s \in S_B^j$ iff $(p_i(s) = 0) \wedge (k_{j-1} \leq i \leq k_j)$. The degree of each p_i is L, (note that given S_B^j, there are $|\mathbb{F}|-1$ possible choices for each p_i). We define the following expression: $P(x_1, \ldots, x_r) := \mathcal{E}(p_1, \ldots, p_k)(x_1, \ldots, x_k) = \sum_{j=1}^{v} \prod_{i=k_{j-1}+1}^{k_j} p_i(x_i)$ (where $k_0 := 0$). Note that $\mathcal{E}$ is 1-bound. Following the type-1 protocol, A securely computes $P(\alpha_1, \ldots, \alpha_k)$. The lists' intersection predicate is $[P(\alpha_1, \ldots, \alpha_r) = 0]$.

Theorem 5. *The above scheme computes the lists' intersection predicate with error probability at most* $1/|\mathbb{F}|$ *(error probability 0, in the case* $v = 1$*).*

Note two interesting special cases: (1) when v=1, A merely checks whether $S_A \cap S_B \neq \emptyset$. If they are disjoint A does not gain any additional information, and if they have common elements A does not obtain a witness. (2) If $\forall j\, (|S_A^j| = 1) \wedge (S_B^j = S_B)$, A checks whether $S_A \subseteq S_B$ (where $S_A := \cup_j S_A^j$)). If this is not the case A does not gain any information about S_B.

An application of the above is the Settlement Escrows Problem [GM95, BN96]: A (the buyer) and B (the seller) negotiate in some fixed price range $[1, \ldots, N]$. B will accept any offer over p_B and A will give at most p_A. A and B wish to know whether p_A and p_B "cross" i.e. $p_A \geq p_B$. Traditionally this problem is solved by revealing the prices to a third party (escrow). Using the lists' intersection predicate scheme twice with $v = 1$, and $S_A = \{1, \ldots, p_A\}$ and $S_B = \{p_B, \ldots, N\}$ each player checks whether there is a cross, without a third party.

7 Other Applications

SGPEs can capture a variety of other "oblivious" interactions between two players. In the full version we present more applications of our construction for type-1 and type-2 games such as *Oblivious Negotiations*, *Oblivious Bargaining*, *Committing to Large Files* and *Oblivious Scoring*.

Acknowledgments. We would like to thank Dan Boneh and Yuval Ishai for their helpful suggestions.

References

[Ber68] Elwyn R. Berlekamp, *Algebraic Coding Theory.* McGraw-Hill, 1968.

[BW86] Elwyn R. Berlekamp and L. Welch, *Error Correction of Algebraic Block Codes.* U.S. Patent, Number 4,633,470 1986.

[BN00] Daniel Bleichenbacher and Phong Nguyen, *Noisy Polynomial Interpolation and Noisy Chinese Remaindering.* In the Proceedings of EUROCRYPT2000, Lecture Notes in Computer Science, Springer, 2000.

[BN96] Adam M. Brandeburger and Barry J. Nalebuff, *Co-opetition*, Doubleday Publications, 1996.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler, *Computationally Private Information Retrieval with Polylogarithmic Communication*, In the Proceedings of EUROCRYPT '99, Lecture Notes in Computer Science, Springer, 1999.

[CG97] Benny Chor and Niv Gilboa, *Computationally Private Information Retrieval*, In the Proceedings of the 29th ACM Symposium on the Theory of Computing, 1997.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan, *Private Information Retrieval*, In the Proceedings of the 36th Annual Symposium on Foundations of Computer Science, 1995.

[GIKM98] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin, *Protecting Data Privacy in Private Information Retrieval Schemes*, In the Proceedings of the 30th ACM Symposium on the Theory of Computing, 1998.

[GM95] Robert H. Gertner and Geoffrey P. Miller, *Settlement Escrows*, Journal of Legal Studies, Vol. 24, pp.87-122, 1995.

[Gol97] S. Goldwasser, *Multi-party computations: Past and present.* In PODC'97, pages 1–6. invited talk.

[GS98] Venkatesan Guruswami and Madhu Sudan, *Improved Decoding of Reed-Solomon and Algebraic-Geometric Codes.* In the Proceedings of the 39th Annual Symposium on Foundations of Computer Science, 1998.

[GMW87] Oded Goldreich, Silvio Micali and Avi Wigderson, *How to Play any Mental Game — A Completeness Theorem for Protocols with Honest Majority.* In the Proceedings of the 19th ACM Symposium on the Theory of Computing, 1987.

[GSR95] Oded Goldreich, Madhu Sudan and Ronitt Rubinfeld, *Learning Polynomials with Queries: The Highly Noisy Case.* In the Proceedings of the 36th Annual Symposium on Foundations of Computer Science, 1995.

[KY01] Aggelos Kiayias and Moti Yung, *Computationally Perfect Symmetric Encryption based on Polynomial Reconstruction*, manuscript, 2001.

[Kil90] Joe Kilian, *Use of Randomness in Algorithms and Protocols.* MIT Press, Cambridge, Massachusetts 1990.

[KO97] Eyal Kushilevitz amd Rafail Ostrovsky, *Replication is not Needed: Single Database, Computationally-Private Information Retrieval*, In the Proceedings of the 38th Annual Symposium on Foundations of Computer Science, 1997.

[MS77] F. J. MacWilliams and N. Sloane, *The Theory of Error Correcting Codes.* North Holland, Amsterdam, 1977.

[NP99] Moni Naor and Benny Pinkas, *Oblivious Transfer and Polynomial Evaluation.* In the Proceedings of the 31th ACM Symposium on the Theory of Computing, 1999.

[Sud97] Madhu Sudan, *Decoding of Reed Solomon Codes beyond the Error-Correction Bound.* Journal of Complexity 13(1), pp. 180–193, 1997.

[Yao86] Andrew C. Yao, *How to Generate and Exchange Secrets.* In the Proceedings of the 27th Annual Symposium on Foundations of Computer Science, 1986.

On the Completeness of Arbitrary Selection Strategies for Paramodulation*

Miquel Bofill[1] and Guillem Godoy[2]

[1] Universitat de Girona, Dept. IMA,
Lluís Santaló s/n, 17071 Girona, Spain
mbofill@ima.udg.es
[2] Technical University of Catalonia, Dept. LSI,
Jordi Girona 1, 08034 Barcelona, Spain
Phone/fax: +34-934017295/7014
ggodoy@lsi.upc.es

Abstract. A crucial way for reducing the search space in automated deduction are the so-called *selection strategies*: in each clause, the subset of *selected* literals are the only ones involved in inferences.
For first-order Horn clauses without equality, resolution is complete with an arbitrary selection of one single literal in each clause [dN96].
For Horn clauses with built-in equality, i.e., paramodulation-based inference systems, the situation is far more complex. Here we show that if a paramodulation-based inference system is complete with eager selection of negative equations and, moreover, it is compatible with equality constraint inheritance, then it is complete with arbitrary selection strategies. A first important application of this result is the one for paramodulation wrt. non-monotonic orderings, which was left open in [BGNR99].

Keywords: automated deduction.

1 Introduction

Deduction with equality is fundamental in mathematics, logics and many applications of formal methods in computer science. During the last two decades this field has importantly progressed through new Knuth-Bendix-like completion techniques and their extensions to ordered paramodulation for first-order clauses. These techniques have lead to important results on theorem proving in first-order logic with equality [HR91,BDH86,BD94,BG94] (that have been applied to state-of-the-art theorem provers like Spass [Wei97]), results on logic-based complexity and decidability analysis [BG96,Nie98], on deduction with *constrained clauses* [KKR90,NR95], and on many other applications like inductive theorem proving,

* Both authors are partially supported by the ESPRIT Basic Research Action CCL-II, ref. WG # 22457. and the Spanish CICYT project HEMOSS ref. TIC98-0949-C02-01. The second author is supported by Departament d'Universitats, Recerca i Societat de la Informació de la Generalitat de Catalunya. A version of this paper with all proofs is available from www.lsi.upc.es/~ggodoy.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 951–962, 2001.

symbolic constraint solving, or equational-logic programming (see [NR01] for a recent survey).

A crucial way for reducing the search space in automated deduction are the so-called *selection strategies*. In such strategies the possible inferences between clauses are restricted to the ones involving *selected* literals. This selection can be done in several different ways. Well-known examples of selection strategies are the *maximal* (or *ordered*) strategies for a given atom ordering. For example, in a maximal resolution strategy, a (ground) inference between $A \vee C$ and $\neg A \vee D$ is performed only if A is larger in the given atom ordering than all other atoms in C and D. Another well-known selection strategy is the so-called *eager* negative selection strategy, where in each clause a single negative literal is selected whenever there is any. This leads to the so-called *positive* strategies (positive unit strategies in the Horn case) because always the left premise of each (resolution or paramodulation) inference is a positive (unit) clause. These strategies are usually easier to prove complete, but sometimes they are not very efficient, because, roughly speaking, one enumerates all solutions of its conditions before using the positive information of a clause (as discussed in [Der91]).

Another well-known way for reducing the search space is constraint inheritance. The semantics of a clause C with a constraint T, written $C \mid T$, is simply the set of all ground instances $C\sigma$ of C such that σ is a solution of T. For example, if $=$ denotes syntactic equality of terms, the constrained clause $P(x) \mid x = f(y)$ denotes all ground atoms of the form $P(f(t))$. Constraints are closely related to *basic* strategies [BGLS95,NR95], where no inferences have to be computed on subterms generated in unifiers of ancestor inference steps. In fact, an inference system with a basic strategy can be modeled as working with constrained clauses, inheriting them and simply storing the unification restrictions in the constraint part (i.e., without applying the constraints to the clause part).

For first-order Horn clauses without equality, resolution is complete with an arbitrary selection of one single literal in each clause ([dN96], Theorem 6.7.4). For Horn clauses with built-in equality, i.e., paramodulation-based inference systems, the situation is far more complex. In [Lyn97] some positive and negative results are given for the case where a total *reduction* (well-founded, monotonic) ordering on ground terms is given. Then arbitrary selection strategies are compatible with superposition[1] (that is, paramodulation involving only maximal sides of equations). Also conditions for eliminating *redundant* clauses are given in [Lyn97], and counter examples indicating the limitations for doing so. For example, in certain circumstances the elimination of tautologies can lead to incompleteness.

Here we show a more general result for Horn clauses with equality, namely that, if a paramodulation-based inference system[2] is complete with eager selection of negative equations and, moreover, it is compatible with equality con-

[1] About two years ago we contacted the author of [Lyn97] about some severe flaws in his completeness proofs, which he agreed to fix, although to our knowledge this has not happened yet.

[2] That is, with inference rules for paramodulation and equality resolution, possibly with constraint inheritance, see [NR01].

straint inheritance (like, in particular, it happens for superposition), then it is complete with arbitrary selection strategies.

Our completeness result is based on transformations of proof trees. Its generality allows us to obtain directly the completeness of arbitrary selection strategies for other inference systems, apart from the one of superposition with total reduction orderings. A first important application of our result is the one for paramodulation with non-monotonic orderings of [BGNR99], where the completeness of strategies different from eager negative selection was left open. There, techniques for dropping the monotonicity requirement were introduced, with the only properties required for the ordering being well-foundedness and the subterm property. However, the inference system of [BGNR99] still required the eager selection of negative equations. In Section 4 we show that those results are compatible with equality constraint inheritance and hence with the *basic* strategy, thus further restricting the search space. Therefore, our transformation method is applicable, and we obtain the completeness of the same inference system but with arbitrary selection strategies.

The structure of the paper is the following. In section 2 we give some basic notions on terms, equations and clauses. In section 3 we present our transformation method for proofs, and our main result on completeness of arbitrary selection strategies. In section 4 we apply this new technique to the case of the inference system of [BGNR99]. Finally, in section 5 we discuss some possible extensions of our technique, giving counter examples indicating the limitations for some of the extensions.

2 Basic Notions

We use the standard definitions of [DJ90]: $T(\mathcal{F}, \mathcal{X})$ ($T(\mathcal{F})$) is the set of (ground) terms over $\mathcal{F}$, the subterm of t at *position* p is denoted $t|_p$, the result of replacing $t|_p$ by s in t is denoted $t[s]_p$, and syntactic equality of terms is denoted by $\equiv$.

A *substitution* σ is a mapping from variables to terms. It can be extended to a function from terms to terms (atoms, clauses) in the usual way: using a postfix notation, $t\sigma$ denotes the result of simultaneously replacing in t every $x \in Dom(\sigma)$ by $x\sigma$. A substitution σ is *ground* if its range is $\mathcal{T}(\mathcal{F})$. Here we suppose that ground substitutions σ are also *grounding*: $t\sigma$ is ground for all t.

An *equation* is a multiset $\{s, t\}$, denoted $s \simeq t$ or, equivalently, $t \simeq s$.

A first-order clause is a pair of finite multisets of equations Γ (the *antecedent*) and Δ (the *succedent*), denoted by $\Gamma \to \Delta$. The empty clause $\Box$ is a clause where both Γ and Δ are empty. For dealing with non-equality predicates, atoms A can be expressed by equations $A \simeq \mathit{true}$ where *true* is a new symbol.

The semantics of a clause C with a constraint T, written $C \mid T$, is simply the set of all ground instances $C\sigma$ of C such that σ is a solution of T.

A (strict partial) *ordering* on $T(\mathcal{F}, \mathcal{X})$ is an irreflexive transitive relation $\succ$. It is a *reduction* ordering if it is well-founded and monotonic, and stable under substitutions: $s \succ t$ implies $s\sigma \succ t\sigma$ for all substitutions σ. It fulfils the *subterm property* if $\succ \supseteq \rhd$, where $\rhd$ denotes the strict subterm ordering.

3 The Transformation Method

In the following we deal with inference systems that are based on some selection strategy. A selection strategy is a function from ground clauses to non-empty sets of literals, such that the selected literals for a clause appear in the clause. An inference between two ground clauses is allowed only if the literals involved in the inference are selected. As usual, a non-ground inference represents all its ground instances fulfilling the required conditions. In our case, a non-ground inference is allowed only if, for some ground instance of the inference, the involved literals are selected.

Our hypothesis in this section is that we have at hand a paramodulation-based inference system $\mathcal{N}$ for first-order Horn clauses, which is compatible with equality constraint inheritance and complete with a concrete strategy with eager selection of negative equations. Let $\mathcal{N}$ consist of the following inference rules:

paramodulation right:

$$\frac{\rightarrow l \simeq r \mid T_1 \qquad \rightarrow s \simeq t \mid T_2}{\rightarrow s[r]_p \simeq t \mid s|_p = l \wedge T_1 \wedge T_2} \qquad \text{if } s|_p \notin \mathcal{X}$$

paramodulation left:

$$\frac{\rightarrow l \simeq r \mid T_1 \qquad \Gamma, \underline{s \simeq t} \rightarrow \Delta \mid T_2}{\Gamma, s[r]_p \simeq t \rightarrow \Delta \mid s|_p = l \wedge T_1 \wedge T_2} \qquad \text{if } s|_p \notin \mathcal{X}$$

equality resolution:

$$\frac{\Gamma, \underline{s \simeq t} \rightarrow \Delta \mid T}{\Gamma \rightarrow \Delta \mid s = t \wedge T}$$

where the equations written underlined must belong to the set of selected literals. Here $=$ is interpreted as the syntactic equality relation $\equiv$ when dealing with instances. That is, we forbid those instances of the conclusion that correspond to ground inferences between instances of the premises for which the constraints do not hold.

Our aim is to prove completeness of the following inference system $\mathcal{A}$, which is a modification of $\mathcal{N}$ allowing an arbitrary selection strategy where a single arbitrary literal is selected in each ground clause:

paramodulation right:

$$\frac{\Gamma_1 \rightarrow \underline{l \simeq r} \mid T_1 \qquad \Gamma_2 \rightarrow \underline{s \simeq t} \mid T_2}{\Gamma_1, \Gamma_2 \rightarrow s[r]_p \simeq t \mid s|_p = l \wedge T_1 \wedge T_2} \qquad \text{if } s|_p \notin \mathcal{X}$$

paramodulation left:

$$\frac{\Gamma_1 \rightarrow \underline{l \simeq r} \mid T_1 \qquad \Gamma_2, \underline{s \simeq t} \rightarrow \Delta \mid T_2}{\Gamma_1, \Gamma_2, s[r]_p \simeq t \rightarrow \Delta \mid s|_p = l \wedge T_1 \wedge T_2} \qquad \text{if } s|_p \notin \mathcal{X}$$

equality resolution:

$$\frac{\Gamma, \underline{s \simeq t} \to \Delta \mid T}{\Gamma \to \Delta \mid s = t \wedge T}$$

In order to prove the completeness of $\mathcal{A}$ we will proceed as follows. Assume S is a set of constrained clauses that is closed under $\mathcal{A}$. Furthermore, let P be a proof by $\mathcal{N}$ deriving the empty clause from S. Then we will show that if P is non-trivial, i.e., it has more than zero steps, then there exists another proof by $\mathcal{N}$ from S of the empty clause with a smaller number of steps. By induction on this proof transformation process, it follows that the empty clause belongs to S.

Let S be a set of constrained clauses and let $C \mid T$ be a constrained clause that is in the closure of S wrt. $\mathcal{N}$. Then, as usual, the proof by $\mathcal{N}$ of $C \mid T$ from S can be expressed as a tree rooted by $C \mid T$, and whose leaves are in S. Now assume T is satisfiable, and let σ be a ground solution of T. Furthermore, σ can be taken such that its domain contains all variables ocurring in the proof. Therefore we can deal with ground proofs where the constraints are replaced by their solution σ (where a ground substitution σ itself is seen as an equality constraint): by a (ground) $\mathcal{N}$*-proof* P of $C \mid \sigma$ from S we mean a proof tree by $\mathcal{N}$, whose nodes are clauses of the form $D \mid \sigma$, and whose leaves are clauses $D' \mid \sigma$ where $D' \mid T'$ is in S and $\sigma \models T'$. By $steps(P)$ we refer to its number of proof steps (or, equivalently, to its number of non-leaf nodes). The following is an example of an $\mathcal{N}$-proof.

Example 1.

$$\dfrac{\dfrac{}{\to c \simeq a \mid x = a} \qquad \dfrac{\dfrac{}{x \simeq a \to b \simeq c \mid x = a}}{\to b \simeq c \mid x = a}}{\to b \simeq a \mid x = a}$$

When dealing with $\mathcal{N}$-proofs, we will frequently speak about its rightmost leaf ($x \simeq a \to b \simeq c \mid x = a$ in the example), its rightmost inner node ($\to b \simeq c \mid x = a$), its rightmost step (the inference obtaining $\to b \simeq c \mid x = a$ from $x \simeq a \to b \simeq c \mid x = a$), and its rightmost path (the nodes $x \simeq a \to b \simeq c \mid x = a$, $\to b \simeq c \mid x = a$, $\to b \simeq a \mid x = a$).

An $\mathcal{N}$-proof is called *antecedent elimination of* Γ if its rightmost leaf is of the form $\Gamma \to \Delta \mid \sigma$, its root is $\to \Delta \mid \sigma$, and no node on its rightmost path is obtained by a paramodulation-right step. In the given proofs, the substitution part $\mid \sigma$ of the clauses is omitted in order to improve readability.

3.1 Completeness Proof

Lemma 1. (fusion lemma) *Let P_1 and P_2 be two antecedent elimination $\mathcal{N}$-proofs of Γ_1 and Γ_2 respectively.*

Then for an arbitrary Δ, there exists an antecedent elimination $\mathcal{N}$-proof P such that its rightmost leaf is $\Gamma_1, \Gamma_2 \to \Delta$. Moreover, $steps(P) = steps(P_1) + steps(P_2)$, and every non-rightmost leaf of P is a non-rightmost leaf of P_1 or of P_2.

In the following, the $\mathcal{N}$-proof P built as in the previous lemma will be called the *fusion* of P_1 and P_2.

Lemma 2. *(separation lemma) Let P be an antecedent elimination $\mathcal{N}$-proof of Γ_1, Γ_2.*

Then, there exist two antecedent elimination $\mathcal{N}$-proofs P_1 and P_2 of Γ_1 and Γ_2 respectively. Moreover $steps(P) = steps(P_1) + steps(P_2)$, and all the non-rightmost leaves of P_1 or of P_2 are non-rightmost leaves of P.

Lemma 3. *(general fusion lemma) Let P_1 be an antecedent elimination $\mathcal{N}$-proof of Γ_1. Let P_2 be an $\mathcal{N}$-proof of $\rightarrow \Delta$ with rightmost leaf $\Gamma_2 \rightarrow \Delta_2$ (therefore a subtree in its rightmost path is antecedent elimination, i.e. the antecedent Γ_2 is eliminated in P_2, but after this elimination, some paramodulation right inferences can be made on the rightmost path).*

Then there exists an $\mathcal{N}$-proof P of $\rightarrow \Delta$ such that its rightmost leaf is $\Gamma_1, \Gamma_2 \rightarrow \Delta_2$. Moreover $steps(P) = steps(P_1) + steps(P_2)$, and every non-rightmost leaf of P is a non-rightmost leaf of P_1 or of P_2.

Lemma 4. *(general separation lemma) Let P be an $\mathcal{N}$-proof of $\rightarrow \Delta$ such that its rightmost leaf is $\Gamma_1, \Gamma_2 \rightarrow \Delta'$.*

Then, there exist two $\mathcal{N}$-proofs P_1 and P_2 such that P_1 is an antecedent elimination proof of Γ_1 and P_2 is an $\mathcal{N}$-proof of $\rightarrow \Delta$ with rightmost leaf $\Gamma_2 \rightarrow \Delta'$. Moreover $steps(P) = steps(P_1) + steps(P_2)$, and all the non-rightmost leaves of P_1 or of P_2 are non-rightmost leaves of P.

Lemma 5. *Let S be a set of clauses closed under $\mathcal{A}$, and let P be an $\mathcal{N}$-proof of $\rightarrow \Delta$ from S.*

Then there exists an antecedent elimination $\mathcal{N}$-proof P' of $\rightarrow \Delta$ from S whose rightmost leaf is of the form $\Gamma \rightarrow \Delta$, where $steps(P') \leq steps(P)$ and, if Γ is non-empty, then Δ is non-empty and its equation is the selected one in $\Gamma \rightarrow \Delta$ by $\mathcal{A}$.

Proof. We will proceed by induction on $steps(P)$. Let the rightmost leaf of P be of the form $\Gamma_1 \rightarrow \Delta_1$. There are several cases to be considered:

0. If Γ_1 and Δ_1 are both empty, then P has no steps and P' can be P itself.

1. One of the selected equations of $\Gamma_1 \rightarrow \Delta_1$ by $\mathcal{A}$ is Δ_1. We consider two possibilities depending on whether some paramodulation right inference is made or not on the rightmost path of P.

1a. In the case that no paramodulation right inference is made on the rightmost path of P, we have that Δ and Δ_1 coincide and P is antecedent elimination. Therefore the P' we are looking for is directly P.

1b. Suppose now there are some paramodulation right inferences on the rightmost path of P. Then the highest one is of the form:

$$\frac{\rightarrow \Delta_2 \qquad \rightarrow \Delta_1}{\rightarrow \Delta_3}$$

Let P_1 be the subproof rooted by $\rightarrow \Delta_1$. It is an antecedent elimination $\mathcal{N}$-proof. Let P_2 be the subproof rooted by $\rightarrow \Delta_2$. Let P_3 be like P, but where P_1 and P_2 are removed (the rightmost leaf of P is $\rightarrow \Delta_3$). Then we have $steps(P) = steps(P_1)+steps(P_2)+steps(P_3)+1$. By applying induction hypothesis to P_2, we obtain an antecedent elimination $\mathcal{N}$-proof P_2' of $\rightarrow \Delta_2$ from S s.t. its rightmost leaf is of the form $\Gamma_2 \rightarrow \Delta_2$, where Δ_2 is selected by $\mathcal{A}$, and $steps(P_2') \leq steps(P_2)$. If we apply the fusion lemma to P_2' and P_1, we obtain an antecedent elimination $\mathcal{N}$-proof P_4 of $\rightarrow \Delta_3$, where its rightmost leaf is $\Gamma_1, \Gamma_2 \rightarrow \Delta_3$, and $steps(P_4) = steps(P_2') + steps(P_1) \leq steps(P_2) + steps(P_1)$. Now let P_5 be the $\mathcal{N}$-proof formed by P_3, and where above its rightmost leaf we insert P_4. We have $\rightarrow \Delta$ in the root of P_5, and $steps(P_5) = steps(P_3) + steps(P_4) \leq steps(P_3) + steps(P_2) + steps(P_1) < steps(P)$ and the rightmost leaf of P_5 is $\Gamma_2, \Gamma_1 \rightarrow \Delta_3$. Moreover, every non-rightmost leaf of P_5 is from S: all the non-rightmost leaves of P_3 are from S, and, since P_4 is the fusion of P_2' and P_1, then, its non-rightmost leaves are from S too. But also, the rightmost leaf of P_5 is from S, since the next inference is a $\mathcal{A}$-inference from S:

$$\frac{\Gamma_2 \rightarrow \underline{\Delta_2} \qquad \Gamma_1 \rightarrow \underline{\Delta_1}}{\Gamma_2, \Gamma_1 \rightarrow \Delta_3}$$

Then the $\mathcal{N}$-proof P' we are looking for is the one obtained by applying the induction hypothesis to P_5.

2. Assume now that Γ_1 is of the form Γ_{11}, e and e is a selected equation of $\Gamma_1 \rightarrow \Delta_1$ by $\mathcal{A}$. We apply the general separation lemma to P and we obtain two $\mathcal{N}$-proofs P_1 and P_2 s.t. P_1 is an antecedent elimination proof of e and P_2 is an $\mathcal{N}$-proof of $\rightarrow \Delta$ with rightmost leaf $\Gamma_{11} \rightarrow \Delta_1$. Moreover $steps(P) = steps(P_1) + steps(P_2)$, and all the non-rightmost leaves of P_1 or of P_2 are non-rightmost leaves of P.

We distinguish two cases depending on wether the rightmost step of P_1 is an equality resolution or a paramodulation left inference on e:

2a. If it is an equality resolution step, it is of the form:

$$\frac{e \rightarrow}{\rightarrow}$$

Then P_1 consists in only this step, and we have the $\mathcal{A}$-inference

$$\frac{\underline{e}, \Gamma_{11} \rightarrow \Delta_1}{\Gamma_{11} \rightarrow \Delta_1}$$

and hence $\Gamma_{11} \rightarrow \Delta_1$ is in S. Therefore the $\mathcal{N}$-proof P' we are looking for is the one obtained by applying the induction hypothesis to P_2.

2b. If the rightmost step of P_1 is a paramodulation left inference, it is of the form:

$$\frac{\rightarrow \Delta_3 \qquad e \rightarrow}{e' \rightarrow}$$

Let P_3 be the subproof of P_1 rooted by $\rightarrow \Delta_3$. Let P_1' be like P_1 but where the subproof P_3 and the rightmost leaf are removed (the rightmost leaf of P_1' is $e' \rightarrow$). Note that all the non-rightmost leaves of P_1' are clauses from S. We have $steps(P_1) = steps(P_3) + steps(P_1') + 1$. We apply induction hypothesis to P_3, and we obtain an antecedent elimination $\mathcal{N}$-proof P_3' of $\rightarrow \Delta_3$ from S s.t. its rightmost leaf is of the form $\Gamma_3 \rightarrow \Delta_3$, where Δ_3 is selected by $\mathcal{A}$. Now, we apply the fusion lemma to P_3' and P_1', and we get an $\mathcal{N}$-proof P_4 s.t. is antecedent elimination of Γ_3, e', and $steps(P_4) = steps(P_3') + steps(P_1')$. Applying now the general fusion lemma to P_4 and P_2 we obtain an $\mathcal{N}$-proof P_5 s.t. its rightmost leaf is $\Gamma_{11}, e', \Gamma_3 \rightarrow \Delta_1$, $steps(P_5) = steps(P_3') + steps(P_1') + steps(P_2) < steps(P)$ and all the non-rightmost leaves of P_5 are non-rightmost leaves of P_3' or of P_1' or of P_2 and, hence, from S. But also the rightmost leaf of P_5 is a clause from S, since the next inference is an $\mathcal{A}$-inference from S:

$$\frac{\Gamma_3 \rightarrow \underline{\Delta_3} \qquad \Gamma_{11}, \underline{e} \rightarrow \Delta_1}{\Gamma_3, \Gamma_{11}, e' \rightarrow \Delta_1}$$

Therefore the $\mathcal{N}$-proof P' we are looking for is the one obtained by applying the induction hypothesis to P_5. □

Theorem 1. *(Completeness theorem) Let S_0 be an unconstrained set of clauses, and let S be the closure of S_0 under $\mathcal{A}$. If S_0 is unsatisfiable, then $\square \in S$.*

Proof. By completeness of $\mathcal{N}$ there is an $\mathcal{N}$-proof of $\square$ from S. Then, applying lemma 5 to the case where Δ is empty gives us a trivial $\mathcal{N}$-proof of $\square$, since Δ can not be selected by $\mathcal{A}$ and, hence, Γ must be empty. Consequently $\square \in S$. □

4 Application to Paramodulation with Respect to Non-monotonic Orderings

Here we prove the refutational completeness of the inference system presented in [BGNR99] with added equality constraint inheritance, for the case of Horn clauses.

Definition 1. *A* west-ordering *is a well-founded ordering on $T(\mathcal{F})$ that fulfils the subterm property and that is total on $T(\mathcal{F})$ (it is called* west *after well-founded, subterm and total).*

For a given a west ordering $\succ$, the inference system $\mathcal{J}$ for Horn clauses with equality is (selected equations are written underlined):

paramodulation right:

$$\frac{\rightarrow l \simeq r \mid T_1 \qquad \rightarrow s \simeq t \mid T_2}{\rightarrow s[r]_p \simeq t \mid s|_p = l \wedge T_1 \wedge T_2}$$

if $s|_p \notin \mathcal{X}$ and $l\sigma \succ r\sigma$ for some ground substitution σ which is a solution of $s|_p = l \wedge T_1 \wedge T_2$.

paramodulation left:

$$\frac{\rightarrow l \simeq r \mid T_1 \qquad \Gamma, \underline{s \simeq t} \rightarrow \Delta \mid T_2}{\Gamma, s[r]_p \simeq t \rightarrow \Delta \mid s|_p = l \wedge T_1 \wedge T_2}$$

if $s|_p \notin \mathcal{X}$ and $l\sigma \succ r\sigma$ for some ground substitution σ which is a solution of $s|_p = l \wedge T_1 \wedge T_2$.

equality resolution:

$$\frac{\Gamma, \underline{s \simeq t} \rightarrow \Delta \mid T}{\Gamma \rightarrow \Delta \mid s = t \wedge T}$$

Theorem 2. *The inference system $\mathcal{J}$ with equality constraint inheritance is refutationally complete for first-order Horn clauses.*

We can use our result of theorem 1 for proving the completeness of a modification $\mathcal{JA}$ of the inference system $\mathcal{J}$. In $\mathcal{JA}$ any strategy selecting a single (positive or negative) equation in each clause is allowed. Note that this new result is not an immediate consequence of theorem 1. The reason is that for right and left paramodulation there is an ordering restriction $l\sigma \succ r\sigma$, and, for explanatory reasons, we did not consider this kind of restrictions in the definition of $\mathcal{N}$. But in the proof transformation from $\mathcal{N}$ to $\mathcal{A}$ one uses a given substitution σ that satisfies all the constraints. This also holds in the case of a proof tree by $\mathcal{J}$ and, moreover, this σ satisfies all the required restrictions $l\sigma \succ r\sigma$ appearing in it. Hence the transformation process works exactly in the same way.

Theorem 3. *The inference system $\mathcal{JA}$ with equality constraint inheritance is refutationally complete for first-order Horn clauses.*

5 Conclusions

We have shown that if a paramodulation-based inference system is complete with a concrete strategy with eager selection of negative equations and, moreover, it is compatible with equality constraint inheritance, then it is complete with arbitrary selection strategies.

Therefore we have generalized the result in [Lyn97] about refutation completeness of arbitrary selection strategies for superposition. Moreover, the generality of our proof transformation method allows us to obtain directly the completeness of arbitrary selection strategies for other inference systems. We have

shown that the results in [BGNR99] for paramodulation with non-monotonic orderings are compatible with equality constraint inheritance, thus further restricting the search space and allowing arbitrary selection strategies.

We have also generalized, in a sense, the result in [dN96] for Horn clauses *without* equality, about completeness of resolution with an arbitrary selection of one single literal in each clause.

In [BG94] standard methods for proving compatibility with redundancy elimination techniques are given, by which, roughly, a clause is redundant if it follows from smaller clauses. These notions are not applicable to our proof transformation technique. But this is not surprising, since by these standard techniques all tautologies are redundant, which is not the case here. Some kind of tautologies have to be kept in order to preserve completeness in the case of arbitrary selection strategies, as shown by the following counter example from [Lyn97].

Example 2. Suppose we have

1. $\rightarrow P(c, b, b)$
2. $P(c, c, b), P(c, b, c) \rightarrow b \simeq c$
3. $P(x, y, y) \rightarrow P(x, y, x)$
4. $P(x, y, y) \rightarrow P(x, x, y)$
5. $P(c, c, c) \rightarrow$

and assume an ordering such that $b \succ c$. Now, assume we are in the case of an inference system applying superposition, with a selection rule such that the positive literal is selected in each program clause. The conclusion of every inference in this set of clauses is identical to an existing clause or is a tautology of the form $A, \Gamma \rightarrow A$. However, the set of clauses is unsatisfiable. And, moreover, it can be easily seen that the empty clause is generated if tautologies of the form $A, \Gamma \rightarrow A$ are kept.

5.1 Future Work

Our result could be extended in several directions, the most interesting ones being:

- Strategies with non-eager selection of negative equations. We think our proof transformation technique could be adapted to cover also strategies with non-eager selection of negative equations, i.e., proving that if there exists *any* (not necessarily eager negative) complete selection strategy selecting a single literal, then arbitrary selection strategies are complete. A possible way to go could be to first transform such proofs into proofs with eager selection of negative equations.
- Answer computation. In this paper we have focussed on refutation completeness, but we believe that our techniques easily extend to proving the completeness for *answer computation*, thus obtaining similar results for this purpose as the ones of [Lyn97] for total reduction orderings and superposition. The key idea is that solutions σ are preserved during our transformation process.

- Non-equality constraints. Very often it is the case that an inference system works not only with equality constraints but also with other kind of constraints such as, for example, ordering constraints. But, as seen in section 4, the whole transformation process can be done in the same way if there exists some substitution σ that satisfies all the constraints. Note that this is a reasonable assumption for many inference systems since, in our transformation, the equations involved in the inferences of the proof by $\mathcal{A}$ are the same as in the proof by $\mathcal{N}$. It may be interesting to study with which other kind of constraints this transformation method preserves completeness.
- General clauses. Our result is for first-order Horn clauses with equality. We think that it would not be difficult to adapt our proof transformation method to the case of general clauses, provided no *factoring* inferences occur in the proofs. With factoring, incompleteness already appears in the propositional case, as shown by the following counter example from [dN96].

  Example 3. Suppose we have

  1. $\rightarrow p, q$
  2. $\rightarrow q, r$
  3. $\rightarrow r, p$
  4. $p, q \rightarrow$
  5. $q, r \rightarrow$
  6. $r, p \rightarrow$

  This set of clauses is unsatisfiable, and factoring (in this propositional case, elimination of repeated occurrences of positive literals) is needed for obtaining the empty clause. Now, suppose we choose the following arbitrary selection:

  1. $\rightarrow \underline{p}, q$
  2. $\rightarrow \underline{q}, r$
  3. $\rightarrow \underline{r}, p$
  4. $\underline{p}, q \rightarrow$
  5. $\underline{q}, r \rightarrow$
  6. $\underline{r}, p \rightarrow$

  By applying resolution involving only the selected literals we only obtain tautologies of the form $A \rightarrow A$, and after resolving with/on them we get clauses of the initial set. Therefore this is a counter example to the completeness of arbitrary strategies when factoring is required, even if all tautologies are kept.

References

[BD94] Leo Bachmair and Nachum Dershowitz. Equational inference, canonical proofs, and proof orderings. *J. of the Association for Computing Machinery*, 41(2):236–276, February 1994.

[BDH86] Leo Bachmair, Nachum Dershowitz, and Jieh Hsiang. Orderings for equational proofs. In *First IEEE Symposium on Logic in Computer Science (LICS)*, pages 346–357, Cambridge, Massachusetts, USA, 1986.

[BG94] Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[BG96] David Basin and Harald Ganzinger. Complexity Analysis Based on Ordered Resolution. In *Eleventh Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 456–465, New Brunswick, New Jersey, USA, 1996.

[BGLS95] L. Bachmair, H. Ganzinger, Chr. Lynch, and W. Snyder. Basic paramodulation. *Information and Computation*, 121(2):172–192, 1995.

[BGNR99] Miquel Bofill, Guillem Godoy, Robert Nieuwenhuis, and Albert Rubio. Paramodulation with non-monotonic orderings. In *14th IEEE Symposium on Logic in Computer Science (LICS)*, pages 225–233, Trento, Italy, 1999.

[Der91] Nachum Dershowitz. Canonical sets of Horn clauses. In J. Leach Albert, B. Monien, and M. Rodríguez Artalejo, editors, *Proceedings of the Eighteenth International Colloquium on Automata, Languages and Programming (ICALP)*, LNCS 510, pages 267–278, Madrid, Spain, 1991. Springer-Verlag.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 6, pages 244–320. Elsevier Science Publishers B.V., Amsterdam, New York, Oxford, Tokyo, 1990.

[DM79] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Comm. of ACM*, 22(8), 1979.

[dN96] Hans de Nivelle. *Ordering refinements of resolution.* Dissertation, Technische Universiteit Delft, Delft, 1996.

[HR91] J. Hsiang and M Rusinowitch. Proving refutational completeness of theorem proving strategies: the transfinite semantic tree method. *Journal of the ACM*, 38(3):559–587, July 1991.

[KKR90] Claude Kirchner, Hélène Kirchner, and Michaël Rusinowitch. Deduction with symbolic constraints. *Revue Française d'Intelligence Artificielle*, 4(3):9–52, 1990.

[Lyn97] C. Lynch. Oriented equational logic programming is complete. *Journal of Symbolic Computation*, 23(1):23–46, January 1997.

[Nie98] Robert Nieuwenhuis. Decidability and complexity analysis by basic paramodulation. *Information and Computation*, 147:1–21, 1998.

[NR95] Robert Nieuwenhuis and Albert Rubio. Theorem Proving with Ordering and Equality Constrained Clauses. *Journal of Symbolic Computation*, 19(4):321–351, April 1995.

[NR01] Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers and MIT Press(to appear), 2001.

[Wei97] Christoph Weidenbach. SPASS—version 0.49. *Journal of Automated Reasoning*, 18(2):247–252, April 1997.

An Axiomatic Approach to Metareasoning on Nominal Algebras in HOAS*

Furio Honsell, Marino Miculan, and Ivan Scagnetto

Dipartimento di Matematica e Informatica, Università di Udine, Italy
{honsell,miculan,scagnett}@dimi.uniud.it

Abstract. We present a logical framework Υ for reasoning on a very general class of languages featuring binding operators, called *nominal algebras*, presented in higher-order abstract syntax (HOAS). Υ is based on an *axiomatic* syntactic standpoint and it consists of a simple types theory *à la* Church extended with a set of axioms called the *Theory of Contexts*, recursion operators and induction principles. This framework is rather expressive and, most notably, the axioms of the Theory of Contexts allow for a smooth reasoning of schemata in HOAS. An advantage of this framework is that it requires a very low mathematical and logical overhead. Some case studies and comparison with related work are briefly discussed.

Keywords: higher-order abstract syntax, induction, logical frameworks.

Introduction

In recent years there has been growing interest in developing systems for defining and reasoning on languages featuring α-conversion. A very promising line of approach has focused on *Higher-Order Abstract Syntax* (HOAS) [7,16,9]. The gist of this approach is to delegate to type-theoretic metalanguages the machinery for dealing with binders. This approach however has some drawbacks. First of all, being equated to metalanguage variables, object level variables cannot be defined inductively without introducing exotic terms [2,13]. A similar difficulty arises with contexts, which are rendered as functional terms. Reasoning by induction and definition by recursion on object level terms is therefore problematic. Finally, the major virtue of HOAS bites back, in the sense that one looses the possibility of reasoning on the properties which are delegated on the metalanguage, *e.g.* substitution and α-equivalence themselves. Various approaches have been proposed to overcome these problems based on different techniques such as modal types, functor categories, permutation models of ZF, etc. [3,4,8, 6,5,12].

The purpose of this paper is to present in broad generality yet another logical framework for reasoning on systems presented in HOAS, called Υ, based on an *axiomatic* syntactic standpoint. This system stems from the technique originally

* Work partially supported by Italian MURST *TOSCA* project and EC-WG *TYPES*.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 963–978, 2001.

used by the authors in [10] for formally deriving in Coq [11] the metatheory of strong late bisimilarity of the π-calculus as in [15].

Υ consists of a simple types theory *à la* Church extended with a set of axioms called the *Theory of Contexts*, recursion operators and induction principles.

According to our experience, this framework is rather expressive. Higher Order Logic allows for the impredicative definition of many relations, possibly functional; the recursors and induction principles allow for the definition of many important functions and properties over contexts; and most notably the axioms in the Theory of Contexts allow for a smooth handling of schemata in HOAS.

We feel that one of the main advantages of our axiomatic approach, compared to other semantical solutions in the literature [4,6], is that it requires a very low mathematical and logical overhead. We do not need to introduce a new abstraction and concretion operators as in [6], but we can continue to model abstraction with λ-abstraction and instantiation with functional application. Therefore our approach can be easily utilized in existing interactive proof assistants, *e.g.* Coq, without needing any redesign of the system.

Of course there are some tradeoffs. One of the major theoretical problems concerning our method is the consistency of the axioms. It is here that we have to resort to more sophisticated mathematical tools, such as *functor categories*, *à la* Hofmann [8]. These are closer in spirit to [4,6], although not quite so, since our method cannot be construed as a mere axiomatization of a topos, but rather of a special tripos model. The consistency of the particular version of the system Υ tailored to the treatment of the π-calculus is proved in [1].

Another tradeoff concerns functions. Our system does not satisfy the *Axiom of Unique Choice* and hence it is functionally not very expressive. The expressive power however can be recovered using functional relations in place of functions.

In this paper we try also to outline in full generality our methodological protocol (which is the one underpinning [10]), using some well-known languages as running examples.

We feel that the present work can be useful for users of interactive proof assistants (Coq, LEGO, Isabelle [18]), as well as developers of programs/processes/ambients calculi willing to consider higher order notions in their theories, as well as implementors of proof assistants willing to extend their systems with principles for reasoning on schemata.

Synopsis. In Section 1 we introduce the notion of *nominal algebra*, together with some examples. In Section 2 we present the logical framework Υ, with the Theory of Contexts. In Section 3 we discuss first-order and higher-order recursion and induction principles in Υ. Some case studies are briefly presented in Section 4. Conclusions and comparison with related work are in Section 5.

1 Nominal Algebras

In this section we present a rather general class of languages with binders, which we call *nominal algebras*. The notion of *binding signature* [4] can be viewed as a special case. Many languages we encounter in logic and computer science can be

easily viewed as nominal algebras. Languages with infinitely many sorts (such as the simply typed λ-calculus *à la* Church) or polyadic languages (such as the polyadic π-calculus) escape the proposed format. We could have easily extended it, at the expense of a more cumbersome notation, but this would have brought in side issues inessential to our purposes.

Definition 1. *A* names set υ *is an infinite enumerable set of different atomic objects, with a decidable equality. A* names base *is a finite set* $V = \{\upsilon_1, \ldots, \upsilon_k\}$ *of names sets.*

Definition 2. *Let* $V = \{\upsilon_1, \ldots, \upsilon_k\}$ *be a names base, whose elements are ranged over by* υ*. Let* $I = \{\iota_1, \ldots, \iota_m\}$ *be a set of* basic types*, ranged over by* ι*.*

A constructor arity over V, I for ι *is a type* α *of the form* $\tau_1 \times \cdots \times \tau_n \to \iota$*, where* $n \geq 0$ *and for* $i = 1 \ldots n$*, the type* τ_i *is either in* V *or it is of the form* $\tau_i = \upsilon_{i1} \times \cdots \times \upsilon_{im_i} \to \sigma_i$ *where* $\upsilon_{ij} \in V$ *and* $\sigma_i \in I$*. If* $m_i > 0$ *for some* i*, then* α *is said to be a* binding arity*, or to* bind $\upsilon_{i1}, \ldots, \upsilon_{im_i}$ over σ_i*.*

A constructor over V, I for ι *is a typed constant* c^α *where* α *is a constructor arity over* V, I*. If* α *is a binding arity, then* c *is said to be a* binding constructor*, or simply a* binder*.*

A nominal algebra N *is a tuple* $\langle V, I, C\rangle$ *where* V *is a set of names sets,* I *is a set of basic types, and* C *is a set of constructors over* V, I*.*

Example 1. Many languages can be viewed as nominal algebras.

- Untyped λ-calculus: $N_\lambda = \langle\{\upsilon\}, \{\Lambda\}, \{var^{\upsilon\to\Lambda}, \boldsymbol{\lambda}^{(\upsilon\to\Lambda)\to\Lambda}, app^{\Lambda\times\Lambda\to\Lambda}\}\rangle$
- First order logic (FOL): $N_{FOL} = \langle\{\upsilon\}, \{\iota, \phi\}, \{var^{\upsilon\to\iota}, 0^\iota, 1^\iota, +^{\iota\times\iota\to\iota}, =^{\iota\times\iota\to\phi}, \supset^{\phi\times\phi\to\phi}, \forall^{(\upsilon\to\phi)\to\phi}\}\rangle$
- Second Order Logic (SOL): $N_{SOL} = \langle\{\upsilon, \upsilon'\}, \{\iota, \phi\}, \{var^{\upsilon\to\iota}, var'^{\upsilon'\to\phi}, 0^\iota, S^{\iota\to\iota}, =^{\iota\times\iota\to\phi}, \supset^{\phi\times\phi\to\phi}, \forall^{(\upsilon\to\phi)\to\phi}, \Lambda^{(\upsilon'\to\phi)\to\phi}\}\rangle$
- π-calculus: $N_\pi = \langle\{\upsilon\}, \{\iota\}, \{0^\iota, |^{\iota\times\iota\to\iota}, \tau^{\iota\to\iota}, =^{\upsilon\times\upsilon\times\iota\to\iota}, \nu^{(\upsilon\to\iota)\to\iota}, in^{\upsilon\times(\upsilon\to\iota)\to\iota}, out^{\upsilon\times\upsilon\times\iota\to\iota}\}\rangle$

Definition 3. *Let* $N = \langle V, I, C\rangle$ *be a nominal algebra. The* object language generated by N*, denoted by* $\mathcal{L}(N)$ *or simply* $\mathcal{L}$*, is the set of well-typed terms definable using the names in* V *and the constructors in* C*, up-to* α*-equivalence. For* $\iota \in I$*, we denote by* $\mathcal{L}^\iota$ *the subset of* $\mathcal{L}_N$ *of terms of type* ι*.*

A stage (in $V = \{\upsilon_1, \ldots, \upsilon_k\}$) *is a tuple* $X = \langle X_1, \ldots, X_k\rangle$ *such that* $X_i \subset \upsilon_i$ *finite for* $i = 1 \ldots k$*. For* X *a stage in* V*, we denote by* $\mathcal{L}_X$ *(resp.,* $\mathcal{L}^\iota_X$*) the subset of* $\mathcal{L}$ *(resp.,* $\mathcal{L}^\iota$*) of terms with free names in* X*.*

2 The Logical Framework Υ

In this section we present the type theoretic logical system Υ for reasoning formally on nominal algebras. Many details of the underlying type theory are not strictly intrinsic. The machinery that we define in this section could have been based on any sufficiently expressive type theory, e.g., CIC. We picked Church Simple Theory of Types only for simplicity.

$$\frac{-}{\Gamma, x:\tau \vdash_\Sigma x:\tau} \text{(VAR)} \qquad \frac{-}{\Gamma \vdash_\Sigma c:\tau}(c{:}\tau) \in \Sigma_c \text{(CONST)}$$

$$\frac{\Gamma \vdash_\Sigma M:\tau' \to \tau \quad \Gamma \vdash_\Sigma N:\tau'}{\Gamma \vdash_\Sigma MN:\tau} \text{(APP)} \qquad \frac{\Gamma \vdash_\Sigma M:o \quad \Gamma \vdash_\Sigma N:o}{\Gamma \vdash_\Sigma M \Rightarrow N:o} \text{(IMP)}$$

$$\frac{\Gamma, x:\tau' \vdash_\Sigma M:\tau}{\Gamma \vdash_\Sigma \lambda x{:}\tau'.M:\tau' \to \tau} \text{(ABS)} \qquad \frac{\Gamma, x:\tau \vdash_\Sigma M:o}{\Gamma \vdash_\Sigma \forall x{:}\tau.M:o} \text{(FORALL)}$$

Fig. 1. Typing rules.

2.1 The Logical Framework Υ: Terms and Types

The logical framework Υ is a theory of Simple Types/Classical Higher Order Logic *à la* Church (HOL) on a given signature Σ.

A *type signature* Σ_t is a finite list of atomic type symbols $\sigma_1, \ldots, \sigma_n$.

The *simple types* over a type signature Σ_t are ranged over by τ, σ (possibly with indices or apices), and are defined by the following abstract syntax:

$$\tau ::= o \mid \sigma \mid \tau \to \tau \quad \text{where } \sigma \in \Sigma_t$$

For each type there is a countably infinite disjoint set of variables.

A *constant signature* Σ_c is a finite list of constant symbols with simple types $c:\tau_1, \ldots, c_m:\tau_m$. A *signature* Σ consists of a *type signature* Σ_t and a *constant signature* Σ_c.

The *terms* over the signature $\Sigma = \langle \Sigma_c, \Sigma_t \rangle$, ranged over by M, N, P, Q, R (possibly with indices), are defined by the following abstract syntax:

$$M ::= x \mid MN \mid \lambda x{:}\tau.M \mid c \mid M \Rightarrow N \mid \forall x{:}\tau.M \quad \text{where } c^\sigma \in \Sigma_c \text{ for some } \sigma$$

As usual, we denote by $M[N/x]$ capture-avoiding substitution. Terms are identified up-to α-conversion.

Remark 1. Primitive datatypes, such as natural numbers and lists, can be easily added to the metalanguage. For the sake of simplicity, however, we prefer to stick to the simplest theory of terms and types. Nevertheless, occasionally we use natural numbers in the following examples.

A *(typing) context* is a finite set of typing assertions over distinct variables (e.g. $\{x_1:\tau_1, x_2:\tau_2, \ldots, x_n:\tau_n\}$). Typing contexts are ranged over by Γ.

Typing judgements have the form $\Gamma \vdash_\Sigma M:\tau$, and they are inductively defined by the rules in Figure 1.

Terms of type o are the *propositions* of our logic. Terms of type $\tau \to o$ are called *predicates (over τ)*. As usual in HOL, all logical connectives can be defined in terms of $\forall$ and $\Rightarrow$. We list some definitions which will be used in the following:

$$\bot \triangleq \forall P{:}o.P \qquad \exists x{:}\tau.P \triangleq \neg\forall x{:}\tau.\neg p \qquad M =^\tau N \triangleq \forall R{:}\tau \to o.(RM \Rightarrow RN)$$
$$\neg P \triangleq P \Rightarrow \bot \qquad P \vee Q \triangleq \neg P \Rightarrow Q \qquad P \wedge Q \triangleq \neg(P \Rightarrow \neg Q)$$

2.2 Encoding Nominal Algebras in Υ

The theory of Υ is expressive enough to represent faithfully any nominal algebra. The standard encoding methodology is based on the *higher-order abstract syntax* paradigm [7,16], which can be succintely summarized as follows:

- object level names are represented by metalanguage variables;
- contexts are represented by higher-order terms, *i.e.* functions;
- binders are represented by constructors which take functions as arguments;
- contexts instantiation and capture-avoiding substitution are meta-level applications; hence, α-conversion is immediately inherited from the metalanguage.

Let $N = \langle V, I, C\rangle$ be a nominal algebra. The *signature for* N, $\Sigma(N)$, is defined as $\Sigma(N) \triangleq \langle V \cup I, \{c : \tau \mid c^\tau \in C\}\rangle$. Then, for each type $\tau \in V \cup I$ and for X stage in V, we define the *encoding map* ϵ^τ_X as follows:

- for $\upsilon \in V$, let $(n_i)_i$ be an enumeration of υ and $(x_i)_i$ be an enumeration of variables of type υ in Υ. Then, $\epsilon^\upsilon_X(n_i) \triangleq x_i$ for $n_i \in X_i$.
  Without loss of generality, we can identify objects of υ with variables of type υ, so that we can define $\epsilon^\upsilon_X(x) \triangleq x$.
- let $c^\alpha \in C$, where $\alpha = \tau_1 \times \cdots \times \tau_n \to \iota$, $\tau_i = \upsilon_{i1} \times \cdots \times \upsilon_{im_i} \to \tau'_i$ $(m_i \geq 0)$ and $\tau'_i \in V \cup I$. For $i = 1 \ldots n$, let Y_i be the stage whose components are $Y_{il} \triangleq X_l \uplus \{x_{ij} \mid \upsilon_{ij} = \upsilon_l, j = 1 \ldots m_i\}$, for $l = 1 \ldots k$. Let $t_i \in \mathcal{L}^{\tau'_i}_{Y_i}$. Then, we define $\epsilon^\iota(c^\alpha(t_1, \ldots, t_n)) \triangleq (c\ \lambda \boldsymbol{x}_1{:}\boldsymbol{\upsilon}_1.\epsilon^{\tau_1}_{Y_1}(t_1) \ldots \lambda \boldsymbol{x}_n{:}\boldsymbol{\upsilon}_n.\epsilon^{\tau_n}_{Y_n}(t_n))$, where $\lambda \boldsymbol{x}_i{:}\boldsymbol{\upsilon}_i$ is a shorthand for $\lambda x_{i1}{:}\upsilon_{i1} \ldots \lambda x_{im_i}{:}\upsilon_{im_i}$.

The canonical terms of Υ over $\Sigma(N)$ correspond faithfully to $\mathcal{L}(N)$:

Theorem 1 (Adequacy of encoding). *Let X be a stage in V, and let $\Gamma(X) \triangleq \{x : \upsilon_i \mid x \in X_i, i = 1 \ldots n\}$. For each type $\iota \in I$, the map ϵ^ι_X is a compositional bijection between $\mathcal{L}^\iota_X$ and the set of terms in $\beta\eta$-normal form of type ι in the context $\Gamma(X)$.*

Example 2. The nominal algebras of Example 1 can be encoded in Υ as follows:

- Untyped λ-calculus: $\Sigma(N_\lambda)_t = \upsilon, \Lambda$,
  $\Sigma(N_\lambda)_c = var : \upsilon \to \Lambda, \boldsymbol{\lambda} : (\upsilon \to \Lambda) \to \Lambda, app : \Lambda \to \Lambda \to \Lambda$
  For instance, $\epsilon^\Lambda_\emptyset(\boldsymbol{\lambda} x x x) = \boldsymbol{\lambda} x{:}\upsilon.(app\ (var\ x)\ (var\ x))$.
- SOL: $\Sigma(N_{SOL})_t = \upsilon, \upsilon', \iota, \phi$; $\Sigma(N_{SOL})_c = var : \upsilon \to \iota, var' : \upsilon' \to \phi, 0 : \iota, S : \iota \to \iota, =: \iota \to \iota \to \phi, \supset : \phi \to \phi \to \phi, \forall : (\upsilon \to \phi) \to \phi, \Lambda : (\upsilon' \to \phi) \to \phi$
- π-calculus: $\Sigma(N_\pi)_t = \upsilon, \iota$; $\Sigma(N_\pi)_c = 0 : \iota, \tau : \iota \to \iota, | : \iota \to \iota \to \iota, =: \upsilon \to \upsilon \to \iota \to \iota, \nu : (\upsilon \to \iota) \to \iota, in : \upsilon \to (\upsilon \to \iota) \to \iota, out : \upsilon \to \upsilon \to \iota \to \iota$

2.3 The Logical Framework Υ: The Logic

Our framework Υ is a full-blown higher order logic. The *logical derivation* judgement "$\Gamma; \Delta \vdash_\Sigma p$" expresses the fact that p derives from the set of propositions Δ in context Γ. Δ is a set of propositions $p_1, \ldots, p_n$ such that $\Gamma \vdash_\Sigma p_i : o$.

The system for deriving judgements consists of a set of *logical* rules and axioms and a set of axioms representing the *Theory of Contexts*. The logical

$$\frac{\Gamma;\Delta,p\vdash_\Sigma q}{\Gamma;\Delta\vdash_\Sigma p\Rightarrow q}\ (\Rightarrow\text{-I}) \qquad \frac{\Gamma\vdash_\Sigma p:o}{\Gamma;\Delta\vdash_\Sigma p\vee\neg p}\ (\text{LEM})$$

$$\frac{\Gamma;\Delta\vdash_\Sigma p\Rightarrow q \quad \Gamma;\Delta\vdash_\Sigma p}{\Gamma;\Delta\vdash_\Sigma q}\ (\Rightarrow\text{-E}) \qquad \frac{\Gamma,x:\tau\vdash_\Sigma M:\sigma \quad \Gamma\vdash_\Sigma N:\tau}{\Gamma;\Delta\vdash_\Sigma (\lambda x{:}\tau.M)N =^\sigma M[N/x]}\ (\beta)$$

$$\frac{\Gamma,x:\tau;\Delta\vdash_\Sigma p}{\Gamma;\Delta\vdash_\Sigma \forall x{:}\tau.p}\,x\notin FV(\Delta)\ (\forall\text{-I}) \qquad \frac{\Gamma\vdash_\Sigma M:\tau\to\sigma}{\Gamma;\Delta\vdash_\Sigma \lambda x{:}\tau.Mx =^{\tau\to\sigma} M}\,x\notin FV(M)\ (\eta)$$

$$\frac{\Gamma;\Delta\vdash_\Sigma \forall x{:}\tau.p \quad \Gamma\vdash_\Sigma M:\tau}{\Gamma;\Delta\vdash_\Sigma p[M/x]}\ (\forall\text{-E}) \qquad \frac{\Gamma,x:\sigma;\Delta\vdash_\Sigma M =^\tau N}{\Gamma;\Delta\vdash_\Sigma \lambda x{:}\sigma.M =^{\sigma\to\tau} \lambda x{:}\sigma.N}\ (\xi)$$

Fig. 2. Logical rules and axioms.

$$\frac{H_1 \quad \dots \quad H_n}{\Gamma;\Delta\vdash_\Sigma x\notin^\iota_\upsilon (c\ M_1\dots M_n)}\,c^{\tau_1\times\dots\times\tau_n\to\iota}\in C \qquad (\text{Notin}_c)$$

$$\text{where } H_i=\begin{cases}\Gamma;\Delta\vdash_\Sigma \neg(x=^\upsilon M_i) & \text{if } \tau_i=\upsilon\\ \Gamma,\Gamma_i;\Delta,\Delta_i\vdash_\Sigma x\notin^{\iota'}_\upsilon (M_i\ y_1\dots y_{m_i}) & \text{if } \tau_i=\upsilon_{i1}\times\dots\times\upsilon_{im_i}\to\iota'\end{cases}$$

$$\Gamma_i=y_1:\upsilon_{i1},\dots,y_{m_i}:\upsilon_{im_i} \qquad \Delta_i=\{\neg(x=^\upsilon y_j)\mid \upsilon_j=\upsilon, j=1\dots m_i\}$$

Fig. 3. Rules for non-occurrence predicates.

rules and axioms (see Figure 2) consist of a natural deduction-style system for classical higher-order logic, with $\beta\eta\xi$-equality.

Before introducing the Theory of Contexts, we define *non-occurrence* $\notin^\iota_\upsilon$: $\upsilon\to\iota\to o$ for each $\upsilon\in V$ and $\iota\in I$. The intuitive meaning of a proposition "$x\notin^\iota_\upsilon M$" (read "x not in M") is "the name x (of type υ) does not appear free in M (of type ι)." (The index $_\upsilon$ will be dropped, when clear from the context.) Since we have higher-order logic, these predicates can be defined by means of higher-order quantifications and monotone operators over predicates, as in [1, 10]:

$x\notin^\iota_\upsilon M \triangleq \forall P{:}\upsilon\to\iota\to o.(\forall y{:}\upsilon.\forall N{:}\iota.(T_{\notin^\iota_\upsilon}\ P\ y\ N)\Rightarrow(P\ y\ N))\Rightarrow(P\ x\ M)$ where $T_{\notin^\iota_\upsilon}:(\upsilon\to\iota\to o)\to(\upsilon\to\iota\to o)$ is a suitable monotone operator defined on the syntax of the language, i.e., on the constructors in C. An explicit definition of these operators, although straightforward, would be quite cumbersome, especially in the case of languages with mutually defined syntactic sorts. Thus for the sake of simplicity we give an implicit definition of the underlying operators by means of a set of "derivation rules" for $\notin^\iota_\upsilon$, as described in Figure 3. It should be clear, however, that these rules are *derivable* from the impredicative definition of the non-occurrence predicates. From a practical point of view, moreover, a rule-based definition is closer to the approach which would be used in proof assistants featuring inductive predicates, as it has been done in [10] using Coq.

Proposition 1 (Adequacy of $\notin^\tau_\upsilon$). *For all Γ contexts, $(x:\upsilon)\in\Gamma$ and M such that $\Gamma\vdash_\Sigma M:\iota$, we have: $\Gamma;\emptyset\vdash_\Sigma x\notin^\tau_\upsilon M$ iff $x\notin FV(M)$*

Proof. By induction on the derivation ($\Rightarrow$), and on the syntax of M ($\Leftarrow$). □

Non-occurrence predicates can be lifted to contexts, that is terms of type $\upsilon_{i_1}\to\dots\to\upsilon_{i_k}\to\iota$:

$$\frac{\Gamma \vdash_\Sigma P : \iota}{\Gamma; \Delta \vdash_\Sigma \exists x{:}\upsilon.x \notin P} \quad (Unsat_\iota^\upsilon)$$

$$\frac{\Gamma \vdash_\Sigma P : \upsilon \to \tau \quad \Gamma \vdash_\Sigma Q : \upsilon \to \tau \quad \Gamma \vdash_\Sigma x : \upsilon}{\Gamma; \Delta, x \not\in^{\upsilon\to\tau} P, x \not\in^{\upsilon\to\tau} Q, (P\ x) =^\tau (Q\ x) \vdash_\Sigma P =^{\upsilon\to\tau} Q} \quad (Ext_\upsilon^\tau)$$

$$\frac{\Gamma \vdash_\Sigma P : \tau \quad \Gamma \vdash_\Sigma x : \upsilon}{\Gamma; \Delta \vdash_\Sigma \exists Q{:}\upsilon \to \tau.x \not\in^{\upsilon\to\tau} Q \wedge P =^\tau (Q\ x)} \quad (\beta\_exp_\upsilon^\tau)$$

where $\tau = \upsilon_{i_1} \to \cdots \to \upsilon_{i_k} \to \iota$

Fig. 4. Axiom schemata for the Theory of Contexts.

$$x \not\in_\upsilon^{\upsilon\to\tau} M \triangleq \forall y{:}\upsilon.\neg(x =^\upsilon y) \Rightarrow x \not\in_\upsilon^\tau (M\ y)$$
$$x \not\in_\upsilon^{\upsilon'\to\tau} M \triangleq \forall y{:}\upsilon'.x \not\in_\upsilon^\tau (M\ y) \qquad (\upsilon \neq \upsilon')$$

Now we can introduce the second set of axioms (Figure 4). This is the real core of the Theory of Contexts. We do not give any informal motivation since these axioms reflect rather natural properties of contexts, when these are rendered by λ-abstractions. They are crucial for reasoning on terms in higher-order abstract syntax, see Theorem 3 and case studies (Section 4).

Remark 2. Our experience in encoding and formally proving metatheoretical properties of nominal algebras indicates that full classical logic is not strictly needed. Actually we could drop rule LEM in Figure 2 and simply assume that either Leibniz equality over names is decidable or occurrence predicates of names in terms are decidable. Indeed, this is the approach we adopted in [10]. More specifically, the two above axioms are rendered in Υ as follows:

$$\frac{\Gamma \vdash_\Sigma x : \upsilon \quad \Gamma \vdash_\Sigma y : \upsilon}{\Gamma; \Delta \vdash_\Sigma x =^\upsilon y \vee x \neq^\upsilon y} \ (\mathrm{LEM}_{=^\upsilon}) \qquad \frac{\Gamma \vdash_\Sigma x : \upsilon \quad \Gamma \vdash_\Sigma P : \iota}{\Gamma; \Delta \vdash_\Sigma x \not\in_\upsilon^\iota P \vee \neg(x \not\in_\upsilon^\iota P)} \ (\mathrm{LEM}_{\not\in_\upsilon^\iota})$$

It is worth noticing that $\mathrm{LEM}_{=^\upsilon}$ derives directly from $\mathrm{LEM}_{\not\in_\upsilon^\iota}$. On the converse, $\mathrm{LEM}_{\not\in_\upsilon^\iota}$ can be derived from $\mathrm{LEM}_{=^\upsilon}$ using $Unsat_\iota^\upsilon$ and by induction both over plain terms and over contexts, using the principles Ind^ι, $Ind^{\upsilon\to\iota}$ (Section 3.1.)

Remark 3. In [10] the Theory of Contexts is enriched by another rather useful axiom stating the monotonicity of $\not\in_\upsilon^\iota$:

$$\frac{\Gamma \vdash_\Sigma x : \upsilon \quad \Gamma \vdash_\Sigma y : \upsilon \quad \Gamma \vdash_\Sigma p : \upsilon \to \iota}{\Gamma; \Delta, x \not\in_\upsilon^\iota (p\ y) \vdash_\Sigma x \not\in_\upsilon^{\upsilon\to\iota} p} \quad (\mathrm{MON}_{\not\in_\upsilon^\iota})$$

Recently, we discovered that the latter is indeed derivable from $Unsat_\iota^\upsilon$, $\mathrm{LEM}_{\not\in_\upsilon^\iota}$ and Ind^ι. Another possibility of deriving $\mathrm{MON}_{\not\in_\upsilon^\iota}$ is to exploit $Ind^{\upsilon\to\iota}$ without any other axioms, i.e., to reason inductively on the structure of the context p.

2.4 Properties of Υ

One upmost concern is soundness:

Theorem 2. *For all nominal algebras N, the framework Υ over the signature $\Sigma(N)$ is sound, in the sense that for all Γ, it is not the case that $\Gamma; \emptyset \vdash_{\Sigma(N)} \bot$.*

The proof of this theorem relies on the construction of a model based on functor categories, following the approach presented in [8]. The construction is quite complex; for further details, we refer the reader to [1,19].

It would be interesting to investigate and formulate more expressive soundness results, which could substantiate the intended meaning of our framework, possibly based on this model. In this paper we discuss solely an important property of Υ which motivates the very Theory of Contexts and can be viewed as a first result in this direction. Let $\Gamma \vdash_\Sigma p : \upsilon \to o$; since terms are taken up-to α-equivalence, we would like the following two rules to be derivable in Υ:

$$\frac{\Gamma; \Delta \vdash_\Sigma \forall y{:}\upsilon.y \not\in^{\upsilon\to o} p \Rightarrow (p\ y)}{\Gamma; \Delta \vdash_\Sigma \exists y{:}\upsilon.y \not\in^{\upsilon\to o} p \wedge (p\ y)}\ (\forall\exists) \qquad \frac{\Gamma; \Delta \vdash_\Sigma \exists y{:}\upsilon.y \not\in^{\upsilon\to o} p \wedge (p\ y)}{\Gamma; \Delta \vdash_\Sigma \forall y{:}\upsilon.y \not\in^{\upsilon\to o} p \Rightarrow (p\ y)}\ (\exists\forall)$$

Indeed, we have that

Theorem 3. *In Υ: $\forall\exists$ is derivable, and $\exists\forall$ is admissible.*

Proof. (Idea.) For $\forall\exists$: some first-order reasoning and an application of $Unsat_\iota^\upsilon$.

For $\exists\forall$: after some first-order reasoning using Ext_υ^τ, it is easy to see that this rule is implied by the following *fresh renaming* rule:

$$\frac{\Gamma, x:\upsilon; \Delta, x \not\in^{\upsilon\to o} p \vdash_\Sigma (p\ x)}{\Gamma, y:\upsilon; \Delta, y \not\in^{\upsilon\to o} p \vdash_\Sigma (p\ y)} x, y \not\in FV(\Delta) \qquad \text{(Ren)}$$

Rule Ren is admissible in our system, because there is no way for a predicate in Υ to discriminate between two fresh names. Thus, a derivation for $(p\ y)$ can be readily obtained by mimicking that for $(p\ x)$ by replacing every free occurrence of x by y. □

Using Ext_υ^τ and $\beta\_exp_\upsilon^\tau$, one can prove that the rule schema $\exists\forall$ is indeed *derivable* in Υ and many of its extensions, for most specific predicates of interest. This is the case, for instance, of the transition relation and strong (late) bisimilarity of π-calculus in Coq [10].

Rule Ren is only admissible because there can be non-standard notions of "predicates" which do not satisfy rule Ren and still are consistent with respect to the Theory of Contexts. An example can be constructed from a logical framework with a distinct type of names $\bar{\upsilon}$, denoting the sum of all classes in the names base. Then, in a nominal algebra with at least two names sets, it would be possible to define a predicate over $\bar{\upsilon}$ which discriminates "red" bound names from "black" bound names, thus invalidating rule Ren.

The type theory of Υ satisfies all the usual properties of simply typed λ-calculi:

Theorem 4. *Let Γ be a context, M, N be terms and τ, σ be types.*

- *(Uniqueness of type) If $\Gamma \vdash_\Sigma M : \tau$ and $\Gamma \vdash_\Sigma M : \sigma$, then $\tau = \sigma$*

- *(Subject reduction) If* $\Gamma \vdash_\Sigma M : \tau$ *and* $\Gamma; \emptyset \vdash_\Sigma M =^\tau N$*, then* $\Gamma \vdash_\Sigma N : \tau$
- *(Normal form) If* $\Gamma \vdash_\Sigma M : \tau$*, then there exists* N *in canonical form such that* $\Gamma; \emptyset \vdash_\Sigma M =^\tau N$
- *(Church-Rosser) If* $\Gamma; \Delta \vdash_\Sigma M =^\tau N_1$ *and* $\Gamma; \Delta \vdash_\Sigma M =^\tau N_2$ *then there exists* N *such that* $\Gamma; \Delta \vdash_\Sigma N_1 =^\tau N$ *and* $\Gamma; \Delta \vdash_\Sigma N_2 =^\tau N$

3 Induction and Recursion in Υ

In this section we face the problem of defining relations and functions by structural induction and recursion, in Υ, and in particular over higher-order types. This very possibility is one of the main properties of the frameworks in [6,5,4].

As originally pointed out in [8] for the case of λ-calculus, a rather surprising tradeoff of our natural framework for treating contexts is the following:

Proposition 2. *The Axiom of Unique Choice*

$$\frac{\Gamma \vdash R : \sigma \to \tau \to o \quad \Gamma, a : \sigma; \Delta \vdash \exists! b : \tau.(R\ a\ b)}{\Gamma; \Delta \vdash \exists f : \sigma \to \tau. \forall a : \sigma.(R\ a\ (f\ a))} \qquad (\mathrm{AC}!_{\sigma,\tau})$$

is inconsistent with the Theory of Contexts.

Proof. Let us consider the case of the π-calculus encoding (see N_π in Example 1), then, by $Unsat^\upsilon_\iota$, we can infer the existence of two fresh names u', v'; hence, we can define the term $R \triangleq \lambda u : \upsilon.\lambda q : \iota.\lambda x : \upsilon.\lambda p : \upsilon.(x =^\upsilon u \wedge p =^\iota 0) \vee (\neg x =^\upsilon u \wedge p =^\iota q)$. It is easy to show that, for all $p' : \iota$, $(R\ u'\ p') : \upsilon \to \iota \to o$ is a functional binary relation. At this point we can prove, by means of Ext^ι_υ and $\mathrm{AC}!_{\upsilon,\iota}$, that the proposition $\forall p : \iota.p =^\iota 0$ holds; Indeed, from $\mathrm{AC}!_{\upsilon,\iota}$ we can deduce the existence of a function $f : \upsilon \to \iota$ such that, for all $x : \upsilon$, $((R\ u'\ p)\ x\ (f\ x))$ holds. Hence, by Ext^ι_υ, we can prove that $f =^{\upsilon\to\iota} \lambda x : \upsilon.p$ because for any fresh name w we have that $(f\ w) =^\iota p$. Then we have that, for all names y, $(f\ y) =^\iota ((\lambda x : \upsilon.p)\ y) =^\iota p$ holds, whence the thesis, since $(f\ u') = 0$.

At this point the contradiction follows because, as a special case, we have that $0|0 = 0$ while ι is an inductive type (the constructors are injective). □

As a consequence of Proposition 2, there are (recursive) functions which cannot be defined as such in Υ, but which can nevertheless be described as functional (inductively defined) relations. We will elaborate more on this in Remark 4.

A rather powerful theory of recursion can nonetheless be obtained also within Υ, following Hofmann [8], exploiting in particular the initial algebra property of the tripos model [1].

3.1 First-Order and Higher-Order Induction

The usual induction principles can be extended naturally to terms possibly containing higher-order terms. In Figure 5 we give the definition of a induction schema for any nominal algebra. In case terms of a syntactic category contain terms of other categories, this scheme turns out to be a mutual induction principle. In this case several inductive properties, one for each syntactic category, are mutually defined. This schema is a direct generalization of those obtained in Coq by Scheme ... Induction [11].

$$\frac{\{\Gamma, \Gamma_c; \Delta, \Delta_c \vdash_\Sigma (P_\iota\ (c\ x_1 \dots x_n)) \mid c^{\tau_1 \times \dots \times \tau_n \to \iota} \in C\}}{\Gamma; \Delta \vdash_\Sigma \forall x{:}\iota.(P_\iota\ x)} \quad (Ind^\iota)$$

where $\tau_i = \upsilon_{i1} \times \dots \times \upsilon_{im_i} \to \iota_i$

$$\Gamma_c \triangleq x_1 : \tau_1', \dots, x_n : \tau_n' \qquad (\tau_i' = \upsilon_{i1} \to \dots \to \upsilon_{im_i} \to \iota_i)$$

$$\Delta_c \triangleq \{\forall y_1{:}\upsilon_{i1}. \dots . \forall y_{m_i}{:}\upsilon_{im_i}.(P_{\iota_i}\ (x_i\ y_1 \dots y_{m_i})) \mid i = 1, \dots, n, \tau_i \notin V\}$$

Fig. 5. First-order induction principle.

$$\frac{\{\Gamma, \Gamma_c; \Delta, \Delta_c \vdash_\Sigma (P_{\upsilon\to\iota}\ \lambda x{:}\upsilon.(c\ t_1 \dots t_n)) \mid c^{\tau_1 \times \dots \times \tau_n \to \iota} \in C\}}{\Gamma; \Delta \vdash_\Sigma \forall M{:}\upsilon \to \iota.(P_{\upsilon\to\iota}\ M)} \quad (Ind^{\upsilon\to\iota})$$

where $\tau_i = \upsilon_{i1} \times \dots \times \upsilon_{im_i} \to \iota_i$ and

$$\Gamma_c \triangleq \{M_i : \upsilon \to \tau_i' \mid \tau_i \notin V\} \cup \{x_i : \tau_i \mid \tau_i \in V\} \quad (\tau_i' = \upsilon_{i1} \to \dots \to \upsilon_{im_i} \to \iota_i)$$

$$\Delta_c \triangleq \{\forall y_1{:}\upsilon_{i1} \dots \forall y_{m_i}{:}\upsilon_{im_i}.(P_{\upsilon\to\iota_i}\ \lambda x{:}\upsilon.(M_i\ x\ y_1 \dots y_{m_i})) \mid i = 1, \dots, n, \tau_i \notin V\}$$

$$t_i \triangleq \begin{cases} (M_i\ x) & \text{if } \tau_i \notin V \\ x_i & \text{if } \tau_i \in V, \tau_i = \upsilon \\ x_i \text{ or } x & \text{if } \tau_i = \upsilon \end{cases}$$

Fig. 6. Higher-order induction principle.

Example 3. The induction principle over terms of π-calculus of the previous examples is the following:

$$\frac{\begin{array}{c}\Gamma; \Delta \vdash (P_\iota\ 0) \qquad \Gamma, x : \iota; \Delta, (P_\iota\ x) \vdash (P_\iota\ (\tau x)) \\ \Gamma, x_1 : \iota, x_2 : \iota; \Delta, (P_\iota\ x_1), (P_\iota\ x_2) \vdash (P_\iota\ (x_1 | x_2)) \\ \Gamma, x_1 : \upsilon, x_2 : \upsilon, x_3 : \iota; \Delta, (P_\iota\ x_3) \vdash (P_\iota\ [x_1 = x_2]x_3) \\ \Gamma, x : \upsilon \to \iota; \Delta, \forall y{:}\upsilon.(P_\iota\ (x\ y)) \vdash (P_\iota\ \nu x) \\ \Gamma, x_1 : \upsilon, x_2 : \upsilon \to \iota; \Delta, \forall y{:}\upsilon.(P_\iota\ (x_2\ y)) \vdash (P_\iota\ (in\ x_1\ x_2)) \\ \Gamma, x_1 : \upsilon, x_2 : \upsilon, x_3 : \iota; \Delta, (P_\iota\ x_3) \vdash (P_\iota\ (out\ x_1\ x_2\ x_3))\end{array}}{\Gamma; \Delta \vdash \forall x{:}\iota.(P_\iota\ x)}$$

The induction principles can be consistently axiomatized also for higher-order terms, i.e. contexts. For the sake of simplicity, we present the general forms of induction principles for simple contexts, that is terms of type $\upsilon \to \iota$ (Figure 6). Nevertheless, the principles can be generalized to any number of abstractions; see [1] for an example and for the semantic justification.

Notice that the terms t_i's are non deterministically defined. In the case that τ_i is exactly υ, both the "x_i" and "x" cases apply. This means that two premises, one for each possible case, have to be taken. Hence, for each constructor c of type $\tau_1 \times \dots \times \tau_n \to \iota$, if υ occurs k times in the sequence $\tau_1, \dots, \tau_n$, then in the rule there are 2^k premises. The following example should make this clear.

$$\frac{F^\tau \text{ elimination scheme over } \tau \text{ in } \Gamma}{\Gamma \vdash_\Sigma \hat{F}^\tau_\iota : \iota \to \tau} \qquad (\hat{F}^\tau_\iota)$$

$$\frac{\Gamma \vdash_\Sigma \hat{F}^\tau_\iota : \iota \to \tau \quad \Gamma \vdash_\Sigma t_1{:}\tau_1 \quad \dots \quad \Gamma \vdash_\Sigma t_n{:}\tau_n}{\Gamma; \Delta \vdash_\Sigma (\hat{F}^\tau_\iota\ (c\ t_1 \dots t_n)) =^\tau (f_c\ M_1 \dots M_n)} \qquad (\hat{F}^\tau_\iota\_eq_c)$$

where $c^{\tau_1 \times \dots \times \tau_n \to \iota} \in C$, $f_c \in F^\tau$ and for $i = 1 \dots n$:

$$M_i \triangleq \begin{cases} \lambda x_{i1}{:}\upsilon_{i1} \dots \lambda x_{im_i}{:}\upsilon_{im_i}.(\hat{F}^\tau_{\iota_i}\ (t_i\ x_{i1} \dots x_{im_i})) & \text{if } \tau_i = \upsilon_{i1} \times \dots \times \upsilon_{im_i} \to \iota_i \\ t_i & \text{if } \tau_i \in V \end{cases}$$

Fig. 7. First-order recursion typing and equivalence rules

Example 4. The induction principle over contexts of λ-calculus is the following:

$$\frac{\begin{array}{c} \Gamma, x_1 : \upsilon; \Delta \vdash (P\ \lambda x{:}\upsilon.(var\ x_1)) \qquad \Gamma, x_1 : \upsilon; \Delta \vdash (P\ \lambda x{:}\upsilon.(var\ x)) \\ \Gamma, M_1{:}\upsilon \to \Lambda, M_2{:}\upsilon \to \Lambda; \Delta, (P\ M_1), (P\ M_2) \vdash (P\ \lambda x{:}\upsilon.(app\ (M_1\ x)\ (M_2\ x))) \\ \Gamma, M_1 : \upsilon \to \upsilon \to \Lambda; \Delta, \forall y_1{:}\upsilon.(P\ \lambda x{:}\upsilon.(M_1\ x\ y_1)) \vdash (P\ \lambda x{:}\upsilon.(\boldsymbol{\lambda}(M_1\ x))) \end{array}}{\Gamma; \Delta \vdash \forall M{:}\upsilon \to \Lambda.(P\ M)}$$

This principle is stronger than the one provided by Coq. In fact, Coq does not recognize that $(M_1\ x)$ is structurally smaller than $\lambda x : \upsilon.(\boldsymbol{\lambda}\ (M_1\ x))$.

3.2 First-Order and Higher-Order Recursion

Despite the drawback we mentioned earlier, many interesting functions can be defined by recursion anyway. Υ can be extended with *recursion operators* or simply *recursors*, which smoothly generalize the usual ones for plain datatypes to HOAS types, and even higher-order types.

Definition 4. *Let $N = \langle V, I, C \rangle$ be a nominal algebra. Let $\iota \in I$, τ be a type in $\Sigma(N)$, and let Γ be a context. An* elimination scheme over τ (in Γ) *is a family of terms $F^\tau = \{f_c \mid c^\alpha \in C\}$ such that for $c^{\tau_1 \times \dots \times \tau_n \to \iota} \in C$, we have*

$$\Gamma \vdash f_c : \tau'_1 \to \dots \to \tau'_n \to \tau$$

where $\tau'_i \triangleq \begin{cases} \tau_i & \text{if } \tau_i \in V \\ \upsilon_{i1} \to \dots \to \upsilon_{im_i} \to \tau & \text{if } \tau_i = \upsilon_{i1} \times \dots \times \upsilon_{im_i} \to \iota_i \end{cases}$

Let F^τ be an elimination scheme in Γ. For each ι in I, we introduce in the language a new symbol, $\hat{F}^\tau_\iota$, called the *F-defined recursive map over ι*. In Figure 7 we give the typing rules for each recursive map, and the equivalence rules for each constructor. This schema naturally generalizes the usual one to the case of mutually defined recursive maps, and to terms possibly containing higher-order terms. This schema is validated by the model of the Theory of Contexts [1]. This recursion schema allows to define many functions on the syntax of terms, e.g., the function which counts the number of abstractions in λ-terms.

Recursion principles can be consistently axiomatized for higher-order terms, i.e. contexts. For the sake of simplicity, we present the general forms of recursion principles for simple contexts, that is terms of type $\upsilon \to \iota$. Nevertheless,

$$\frac{F^\tau \text{ } \upsilon\text{-elimination scheme over } \tau \text{ in } \Gamma}{\Gamma \vdash_\Sigma \hat{F}^\tau_\iota : (\upsilon \to \iota) \to \tau} \qquad (\hat{F}^\tau_\iota)$$

$$\frac{\Gamma \vdash_\Sigma \hat{F}^\tau_\iota : (\upsilon \to \iota) \to \tau \quad \{\Gamma \vdash_\Sigma t_i : \upsilon \to \tau_i \mid \tau_i \notin V\} \quad \{\Gamma \vdash_\Sigma y_i : \tau_i \mid \tau_i \in V\}}{\Gamma; \Delta \vdash_\Sigma (\hat{F}^\tau_\iota \ \lambda x{:}\upsilon.(c\ N_1 \dots N_n)) =^\tau (f^l_c\ M_1 \dots M_n)} \qquad (\hat{F}^\tau_\iota \_eq^l_c)$$

where $c^{\tau_1 \times \dots \times \tau_n \to \iota} \in C$, $l \in L(\tau_1 \times \dots \times \tau_n \to \iota)$, $f^l_c \in F^\tau$, and for $i = 1 \dots n$:

$$N_i = \begin{cases} (t_i\ x) & \text{if } \tau_i \notin V \\ y_i & \text{if } \tau_i \in V \text{ and } (\tau_i \neq \upsilon \text{ or } l_i = 1) \\ x & \text{if } \tau_i = \upsilon \text{ and } l_i = 0 \end{cases}$$

$$M_i = \begin{cases} \lambda x_{i1}{:}\upsilon_{i1} \dots \lambda x_{im_i}{:}\upsilon_{im_i}.(\hat{F}^\tau_{\iota_i}\ \lambda x{:}\upsilon.(t_i\ x\ x_{i1} \dots x_{im_i})) & \\ & \text{if } \tau_i = \upsilon_{i1} \times \dots \times \upsilon_{im_i} \to \iota_i \\ y_i & \text{if } \tau_i \in V \text{ and } (\tau_i \neq \upsilon \text{ or } l_i = 1) \\ (\mathit{nothing}) & \text{if } \tau_i = \upsilon \text{ and } l_i = 0 \end{cases}$$

Fig. 8. Higher-order recursion typing and equivalence rules.

the principles can be generalized to any number of abstractions; see [1] for an example and for the semantic justification of the principles.

Before giving the definitions and the rules, we need some notation. Let $\alpha = \tau_1 \times \dots \times \tau_n \to \iota$ be an arity of constructor. Let $\upsilon \in V$ and let $1 \leq i_1 \leq \dots \leq i_k \leq n$ $(k \geq 0)$ be the indices such that $\tau_{i_j} = \upsilon$. Let $L(\alpha) \triangleq \{0,1\}^k$ be the set of binary strings of length k, which we call the *labels for* α. (Thus, $|L(\alpha)| = 2^k$.) For $j = 1 \dots k$, the j-th component of a label l is denoted by l_{i_j}, that is it has the same index of the occurrence of υ in $\tau_1 \dots \tau_n$ it refers to. Finally, we denote by $l \bullet (\tau'_1 \to \dots \to \tau'_n \to \tau)$ the type obtained from $\tau'_1 \to \dots \to \tau'_n \to \tau$ by eliminating τ'_{i_j} if $l_{i_j} = 0$.

Definition 5. *Let $\iota \in I$, $\upsilon \in V$, in $\Sigma(N)$, and let Γ be a context. A υ-elimination scheme over τ (in Γ) is a family of terms $F^\tau = \{f^l_c \mid c^\alpha \in C, l \in L(\alpha)\}$ such that for $c^{\tau_1 \times \dots \times \tau_n \to \iota} \in C$ and $l \in L(\tau_1 \times \dots \times \tau_n \to \iota)$, we have*

$$\Gamma \vdash f^l_c : l \bullet (\tau'_1 \to \dots \to \tau'_n \to \tau)$$

where $\tau'_i \triangleq \begin{cases} \upsilon & \text{if } \tau_i = \upsilon \\ \upsilon_{i1} \to \dots \to \upsilon_{im_i} \to \tau & \text{if } \tau_i = \upsilon_{i1} \times \dots \times \upsilon_{im_i} \to \iota_i \end{cases}$

Hence, for each constructor c^α, there are $|L(\alpha)|$ terms in F^τ.

Let F^τ be a υ-elimination scheme in Γ. For each ι in I, we introduce in the language a new symbol, $\hat{F}^\tau_\iota$, called the *F-defined recursive map over $\upsilon \to \iota$*. In Figure 8 we give the typing rules for each recursive map, and the equivalence rules for each term in F^τ. This schema is validated by the categorical model of the Theory of Contexts [1].

Example 5. In the signature $\Sigma(N_\lambda)$, we will define, by higher-order recursion, the substitution $\cdot[N/\cdot]$ for a given λ-term such that $\Gamma \vdash N : \Lambda$. The three sets of

labels are $L(\upsilon \to \Lambda) = \{0,1\}$, $L(\Lambda \to \Lambda \to \Lambda) = L((\upsilon \to \Lambda) \to \Lambda) = \{\langle\rangle\}$. Thus, let $F^{\Lambda} \triangleq \{f^0_{var}, f^1_{var}, f_{app}, f_{\boldsymbol{\lambda}}\}$ where $f^0_{var} \triangleq N$, $f^1_{var} \triangleq var$, $f_{app} \triangleq app$, $f_{\boldsymbol{\lambda}} \triangleq \boldsymbol{\lambda}$. Then, F^{Λ} is a υ-elimination schema in Γ, such that $\Gamma \vdash \hat{F}^{\Lambda}_{\Lambda} : (\upsilon \to \Lambda) \to \Lambda$ and the following are derivable:

$$\frac{-}{\Gamma;\Delta \vdash \hat{F}^{\Lambda}_{\Lambda}(var) =^{\Lambda} N} \qquad \frac{\Gamma \vdash y : \upsilon}{\Gamma;\Delta \vdash \hat{F}^{\Lambda}_{\Lambda}(\lambda x{:}\upsilon.(var\ y)) =^{\Lambda} (var\ y)}$$

$$\frac{\Gamma \vdash M : \upsilon \to \upsilon \to \Lambda}{\Gamma;\Delta \vdash \hat{F}^{\Lambda}_{\Lambda}(\lambda x{:}\upsilon.\boldsymbol{\lambda}(M\ x)) =^{\Lambda} \boldsymbol{\lambda}\lambda x{:}\upsilon.\hat{F}^{\Lambda}_{\Lambda}(M\ x)}$$

$$\frac{\Gamma \vdash M_1 : \upsilon \to \Lambda \quad \Gamma \vdash M_2 : \upsilon \to \Lambda}{\Gamma;\Delta \vdash \hat{F}^{\Lambda}_{\Lambda}(\lambda x{:}\upsilon.(app\ (M_1\ x)\ (M_2\ x)) =^{\Lambda} (app\ \hat{F}^{\Lambda}_{\Lambda}(M_1)\ \hat{F}^{\Lambda}_{\Lambda}(M_2))}$$

Hence, for any M term and x variable, the term $\hat{F}^{\Lambda}_{\Lambda}(\lambda x{:}\upsilon.M)$ is equal to the term obtained from M by replacing every free occurrence of x by N.

Remark 4. There are functions that we cannot define in Υ. The reason is that, Υ does not allow to define functions whose definitions need freshly generated names, since there is no means for generating a "fresh name" at the term level, while we can use $Unsat^{\upsilon}_{\tau}$ for generating fresh names at the logical level. Nevertheless, n-ary functions of this kind can be represented in Υ as $(n+1)$-ary relations, as in the next Example.

Example 6. Let us consider the function $count : \Lambda \to nat$ which takes as argument a term M of type Λ and returns the number of occurrences of free variables occurring in M. The corresponding elimination scheme over nat should be $f_{var} \triangleq \lambda x : \upsilon.1$, $f_{app} \triangleq \lambda n : nat.\lambda n' : nat.n + n'$, $f_{\boldsymbol{\lambda}} \triangleq \lambda g : \upsilon \to nat.(g\ z)\dot{-}1$, where $\dot{-}1$ denotes the predecessor function over natural numbers. However, the above definition cannot be expressed in Υ since the fresh name z, needed in the definition of $f_{\boldsymbol{\lambda}}$, is not definable. We do not have a mechanism working at the level of datatypes for generating fresh names on the spot, like Gabbay and Pitts' *fresh* operator [6]. It is straightforward that, in the presence of such a fresh operator, $f_{\boldsymbol{\lambda}}$ can be defined as $\lambda g : \upsilon \to nat.fresh\ z.(g\ z)\dot{-}1$. However, we can represent $f_{\boldsymbol{\lambda}}$ as a binary relation $R_{\boldsymbol{\lambda}} : (\upsilon \to nat) \to nat \to o$ defined as

$$R_{\boldsymbol{\lambda}}(g,n) \triangleq \exists z : nat.z \not\in^{\upsilon \to nat} g \wedge (g\ z)\dot{-}1 =^{nat} n$$

the existence of the fresh variable z being granted by $Unsat^{\upsilon}_{nat}$. Hence, the *fresh* operator can be mimicked at the logical level by our $Unsat^{\upsilon}_{\iota}$ axiom scheme.

4 Case Studies

In order to prove the usability and expressiveness of our axiomatic Theory of Contexts, we carried out several case studies about metatheoretical reasoning on HOAS-based encodings. The first case study, which indeed yielded the first version of our axiomatization, is the encoding of Milner's π-calculus as presented in [15]. We refer for more details to [10]; here we will only remark that the Theory of Contexts allowed us to formally derive in Coq a substantial chapter of the metatheory of strong late bisimilarity. For instance, the following property

[15, Lemma 6] If $P \nsim Q$ and $w \notin fn(P,Q)$, then $P\{w/x\} \nsim Q\{w/x\}$.

has been formally derived, using all axioms of the Theory of Contexts.

Another substantial case study concerned the untyped and simply typed λ-calculus in Coq [14]. A set of important metatheoretical results has been formally proved by means of the Theory of Contexts, such as determinism and totality of substitution:

For all M, N, x, there exists exactly one M' such that $M' = M[N/x]$.

In this development, substitution has been defined as a relation between contexts and terms, and the proof of totality relies on higher-order induction over contexts. Other properties which have been proved include determinism and totality of big-step semantics, subject reduction and equivalence between small-step and big-step semantics. A similar case study has been carried out for First Order Logic, and on a smaller scale for a λ-calculus with explicit substitutions.

Currently there is work in progress on more complex process algebras, namely the spi calculus and the ambient calculus. These case studies are quite challenging for testing the expressivity of our axiomatization, since they provide a notion of substitution of terms for names, while the original π-calculus only relies on substitution of names for names. Some of the modal logics for ambient calculi are troublesome also because they introduce distinct sets for names and variables.

5 Comparison with Related Work and Concluding Remarks

The Theory of Contexts and Isabelle/HOL. The Theory of Contexts can be used in many different logical frameworks in order to reason on higher-order abstract syntax. A HOAS-based encoding of the syntax of π-calculus processes in Isabelle/HOL is given in [18]. The axioms $MON_{\not\in^\iota_\upsilon}$, Ext^ι_υ and $\beta\_exp^\iota_\upsilon$ are formally derived there from well-formedness predicates. The proof of the monotonicity of the occurrence predicate $\not\in^\iota_\upsilon$ is straightforward, since this is not defined independently, but simply as the negation of $\in^\iota_\upsilon$. It can be formally proved equivalent to the constructive one, however, by means of $Unsat^\upsilon_\iota$ and Ind^ι and $\mathrm{LEM}_{=^\upsilon}$. The proofs of extensionality and β-expansion rely heavily on the fact that Isabelle/HOL implements an extensional equality. These proofs cannot be mimicked in COQ.

The FM approach. Gabbay and Pitts in [6] introduce a system, called FM, with a special quantifier for expressing freshness of names. The intuitive meaning of $\mathcal{N}y.p$ is "p holds for y a fresh name". $\mathcal{N}$ resembles both $\forall$ and $\exists$, and it satisfies the rules:

$$\frac{\Gamma, y\#\boldsymbol{x} \vdash p}{\Gamma \vdash \mathcal{N}y.p} \qquad \frac{\Gamma \vdash \mathcal{N}y.p \quad \Gamma, p, y\#\boldsymbol{x} \vdash q}{\Gamma \vdash q}$$

where $\boldsymbol{x}$ is the "support" of p. In the Theory of Contexts, $\mathcal{N}y.p$ and $y\#\boldsymbol{x}$ can be encoded as follows:

$$\mathcal{N}y.p \triangleq \forall y{:}\upsilon.y \not\in^{\upsilon\to o} (\lambda y{:}\upsilon.p) \Rightarrow p \qquad y\#\boldsymbol{x} \triangleq y \not\in^o p$$

Rules, corresponding to the ones above, can then be easily derived using the Theory of Contexts. The abstraction ($x.a$) and instantiation ($a@x$) operators are taken as primitives in FM. In our approach both can be rendered naturally, using the features of the metalanguage: the first as λ-abstraction, the latter as application. Notice that instantiation in FM is only partially defined, *i.e.*, when x is not in the support of a, *i.e.*, the free variables of a. The *fresh* operator, on the other hand, cannot be encoded at the level of terms. Its uses however can be recovered at the level of predicates, see the paragraph below. It is interesting to notice that this condition has a bearing on the fact that AC! holds in FM.

Correspondingly, suitable adaptations of our Theory of Contexts are validated in the FM. More experiments need to be carried out to verify the adequacy of these translations, and to compare the advantages of using FM versus the Theory of Contexts in, possibly mechanized, proof search.

Programming in Υ. Currently there is a great deal of research on programming languages featuring contexts and binding structures as datatypes [17,12,5]. The term language of Υ could be extended naturally to a functional programming language to this end, possibly adding a fixed point operator. However, in view of Remark 4 this would not be a very expressive language. A much better alternative would be to define separately a programming language whose semantics would be the functional relations in Υ. An operational semantics for this language could be given directly or in a logic programming style. A program logic for this language should be derivable from Υ. More work needs to be done in this direction.

Completeness of the Theory of Contexts. An open question.

References

1. A. Bucalo, M. Hofmann, F. Honsell, M. Miculan, and I. Scagnetto. Using functor categories to explain and justify an axiomatization of variables and schemata in HOAS. In preparation, 2001.
2. J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order syntax in Coq. In *Proc. of TLCA'95*, LNCS 905. Springer-Verlag, 1995.
3. J. Despeyroux, F. Pfenning, and C. Schürmann. Primitive recursion for higher order abstract syntax. Technical Report CMU-CS-96-172, Carnegie Mellon University, September 1996.
4. M. P. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In G. Longo, ed., *Proc. 14th LICS*, pages 193–202. IEEE, 1999.
5. M. J. Gabbay. *A Theory of Inductive Definitions With α-equivalence*. PhD thesis, Trinity College, Cambridge University, 2000.
6. M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax involving binders. In G. Longo, ed., *Proc. 14th LICS*, pages 214–224. IEEE, 1999.
7. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, Jan. 1993.
8. M. Hofmann. Semantical analysis of higher-order abstract syntax. In G. Longo, ed., *Proc. 14th LICS*, pages 204–213. IEEE, 1999.
9. F. Honsell and M. Miculan. A natural deduction approach to dynamic logics. In *Proc. of TYPES'95*, LNCS 1158, pages 165–182. Springer-Verlag, 1996.

10. F. Honsell, M. Miculan, and I. Scagnetto. π-calculus in (co)inductive type theory. *TCS* 253(2):239–285, 2001. First appeared as a talk at TYPES'98 annual workshop.
11. INRIA. *The Coq Proof Assistant*, 2000. `http://coq.inria.fr/doc/main.html` .
12. R. McDowell and D. Miller. A logic for reasoning with higher-order abstract syntax. In *Proc. 12th LICS*. IEEE, 1997.
13. M. Miculan. *Encoding Logical Theories of Programs*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy, Mar. 1997.
14. M. Miculan. Encoding and metareasoning of call-by-name λ-calculus. Available at `http://www.dimi.uniud.it/~miculan/CoqCode/HOAS`, 2000.
15. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Inform. and Comput.*, 100(1):1–77, 1992.
16. F. Pfenning and C. Elliott. Higher-order abstract syntax. In *Proc. of ACM SIGPLAN '88*, pages 199–208, 1988.
17. A. M. Pitts and M. J. Gabbay. A metalanguage for programming with bound names modulo renaming. In *Proc. MPC2000*, LNCS 1837, pages 230–255. Springer, 2000.
18. C. Röckl, D. Hirschkoff, and S. Berghofer. Higher-order abstract syntax with induction in Isabelle/HOL: Formalising the π-calculus and mechanizing the theory of contexts. In *Proc. FOSSACS 2001*, LNCS 2030, pages 359–373. Springer, 2001.
19. I. Scagnetto. *Reasoning on Names In Higher-Order Abstract Syntax*. PhD thesis, Dip. di Matematica e Informatica, Università di Udine, 2002. In preparation.

Knuth-Bendix Constraint Solving Is NP-Complete

Konstantin Korovin and Andrei Voronkov

University of Manchester
{korovin,voronkov}@cs.man.ac.uk

Abstract. We show the NP-completeness of the existential theory of term algebras with the Knuth-Bendix order by giving a nondeterministic polynomial-time algorithm for solving Knuth-Bendix ordering constraints.

1 Introduction

Solving ordering constraints in term algebras with various reduction orders is used in rewriting to prove termination of recursive definitions, and in automated deduction to prune the search space [2,7,13]. Nieuwenhuis [13] connects further progress in automated deduction with constraint-based deduction.

Two kinds of orders are used in automated deduction: the Knuth-Bendix order [8] and various versions of recursive path orders [3]. The Knuth-Bendix order is used in the state-of-the-art theorem provers, for example, E [15], SPASS [16], Vampire [14], and Waldmeister [4]. There is extensive literature on solving recursive path ordering constraints (e.g., [2,6,12,11]). The decidability of Knuth-Bendix ordering constraints was proved only recently in [9]. The algorithm described in that paper shows that the problem belongs to 2-NEXPTIME. It was also shown that the problem is NP-hard by reduction of the solvability of systems of linear Diophantine equations to the solvability of the Knuth-Bendix ordering constraints. In this paper we present a nondeterministic polynomial-time algorithm for solving Knuth-Bendix ordering constraints, and hence show that the problem is NP-complete. As a consequence, we obtain that the existential first-order theory of any term algebra with a Knuth-Bendix order is NP-complete too.

This paper is structured as follows. In Section 2 we define the main notions of this paper. In Section 3 we introduce the notion of isolated form of constraints and show that every constraint can be effectively transformed into an equivalent disjunction of constraints in isolated form. This transformation is represented as a nondeterministic polynomial-time algorithm computing members of this disjunction. After this, it remains to show that solvability of constraints in isolated form can be decided by a nondeterministic polynomial-time algorithm. In Section 4 we present such an algorithm using transformation to systems of linear Diophantine inequalities over the weights of variables. Finally, in Section 5 we complete the proof of the main result.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 979–992, 2001.

The proofs in this paper are rather involved and we had to omit some of them because of the page limit. For the proofs of statements labeled with □ consult [10].

2 Preliminaries

A *signature* is a finite set of function symbols with associated arities. In this paper we assume an arbitrary but fixed signature Σ. *Constants* are function symbols of the arity 0. We assume that Σ contains at least one constant. We denote variables by x, y, z and terms by r, s, t. The set of all ground terms of the signature Σ can be considered as the *term algebra* of this signature, $\mathrm{TA}(\Sigma)$, by defining the interpretation $g^{\mathrm{TA}(\Sigma)}$ of any function symbol g by $g^{\mathrm{TA}(\Sigma)}(t_1, \ldots, t_n) = g(t_1, \ldots, t_n)$. For details see e.g. [5].

Denote the set of natural numbers by $\mathbb{N}$. We call a *weight function* on Σ any function $w : \Sigma \to \mathbb{N}$. A *precedence relation* on Σ is any linear order $\gg$ on Σ.

The definition of the Knuth-Bendix order on $\mathrm{TA}(\Sigma)$ is parametrized by (i) a weight function w on Σ; and (ii) a precedence relation $\gg$ on Σ such that (a) $w(a) > 0$ for every constant a and (b) if f is a unary function symbol and $w(f) = 0$, then f must be the greatest element of Σ w.r.t. $\gg$. These conditions on the weight function ensure that the Knuth-Bendix order is a simplification order total on ground terms (see e.g., [1]). In this paper, f will always denote a unary function symbol of the weight 0.

In the sequel we assume a fixed weight function w on Σ and a fixed precedence relation $\gg$ on Σ. The *weight* of any ground term t, denoted $|t|$, is defined as follows: for any constant c we have $|c| = w(c)$ and for any function symbol g of a positive arity $|g(t_1, \ldots, t_n)| = w(g) + |t_1| + \ldots + |t_n|$.

The *Knuth-Bendix order* on $\mathrm{TA}(\Sigma)$ is the binary relation $\succ$ defined as follows. For any ground terms $t = g(t_1, \ldots, t_n)$ and $s = h(s_1, \ldots, s_k)$ we have $t \succ s$ if one of the following conditions holds:

1. $|t| > |s|$;
2. $|t| = |s|$ and $g \gg h$;
3. $|t| = |s|$, $g = h$ and for some $1 \leq i \leq n$ we have $t_1 = s_1, \ldots, t_{i-1} = s_{i-1}$ and $t_i \succ s_i$.

The main result of this paper is the following

Theorem 2: *The existential first-order theory of any term algebra with the Knuth-Bendix order is NP-complete.*

To prove this, we introduce a notion of Knuth-Bendix ordering constraint and show

Theorem 1: *The problem of solving Knuth-Bendix ordering constraints is NP-complete.*

The proof will be given after a series of lemmas. The idea of the proof is as follows. First, we will make $\mathrm{TA}(\Sigma)$ into a two-sorted structure by adding the sort of natural numbers, and extend its signature by (i) the weight function on ground terms; (ii) the addition function on natural numbers; (iii) the Knuth-Bendix order on ground terms.

Given an existential formula of the first-order theory of a term algebra with the Knuth-Bendix order, we will transform it step by step into an equivalent disjunction of existential formulas of the extended signature. The main aim of these steps is to replace all occurrences of $\succ$ by linear Diophantine inequalities on the weights of variables. After such a transformation we will obtain existential formulas consisting of linear Diophantine inequalities on the weight of variables plus statements expressing that, for some fixed natural number N, there exists at least N terms of the same weight as $|x|$, where x is a variable. We will then show how these statements can be expressed using systems of linear Diophantine inequalities on the weights of variables and then we can use a nondeterministic polynomial-time algorithm to solve obtained systems of linear Diophantine inequalities.

We denote by $\mathrm{TA}^+(\Sigma)$ the following structure with two sorts: the *term algebra sort* and the *arithmetical sort*. The domains of the term algebra sort and the arithmetical sort are the sets of ground terms of Σ and natural numbers, respectively. The signature of $\mathrm{TA}^+(\Sigma)$ consists (i) of all symbols of Σ interpreted as in $\mathrm{TA}(\Sigma)$, (ii) symbols $0, 1, >, +$ having their conventional interpretation over natural numbers, (iii) the binary relation symbol $\succ$ on the term algebra sort, interpreted as the Knuth-Bendix order, (iv) the unary function symbol $|\ldots|$, interpreted as the weight function. When we need to distinguish the equality $=$ on the term algebra sort from the equality on the arithmetical sort, we denote the former by $=_{\mathrm{TA}}$, and the latter by $=_{\mathbb{N}}$.

We will prove that the existential theory of $\mathrm{TA}^+(\Sigma)$ is in NP, from which the fact that the existential theory of any term algebra with the Knuth-Bendix order belongs to NP follows immediately. We consider *satisfiability* and *equivalence* of formulas with respect to the structure $\mathrm{TA}^+(\Sigma)$. We call a *constraint* in the language of $\mathrm{TA}^+(\Sigma)$ any conjunction of atomic formulas of this language.

Proposition 1. *The existential theory of* $\mathrm{TA}^+(\Sigma)$ *is in NP if and only if so is the constraint satisfiability problem.*

A *substitution* is a mapping from a set of variables to the set of terms. A substitution θ is called *grounding* for an expression C (i.e., term or constraint) if for every variable x occurring in C the term $\theta(x)$ is ground. Let θ be a substitution grounding for an expression C. We denote by $C\theta$ the expression obtained from C by replacing in it every variable x by $\theta(x)$. A substitution θ is called a *solution* to a constraint C if θ is grounding for C and $C\theta$ is valid.

In the sequel we will often replace a constraint $C(\bar{x})$ by a formula $A(\bar{x}, \bar{y})$ containing extra variables $\bar{y}$ and say that they are "equivalent". By this we mean that $\mathrm{TA}^+(\Sigma) \models \forall \bar{x}(C(\bar{x}) \leftrightarrow \exists \bar{y} A(\bar{x}, \bar{y}))$. In other words, the solutions to C are exactly the solutions to A projected on $\bar{x}$.

3 Isolated Forms

We are interested not only in satisfiability of constraints, but also in their solutions. Our algorithm will consist of equivalence-preserving transformation steps. When the signature contains no unary function symbol of the weight 0, the transformation will preserve equivalence in the following strong sense. At each step, given a constraint $C(\bar{x})$, we transform it into constraints $C_1(\bar{x},\bar{y}),\ldots,C_n(\bar{x},\bar{y})$ such that for every sequence of ground terms $\bar{t}$, the constraint $C(\bar{t})$ holds if and only if there exist k and a sequence of ground terms $\bar{s}$ such that $C_k(\bar{t},\bar{s})$ holds. Moreover this transformations will be presented as a nondeterministic polynomial-time algorithm which computes on every branch some $C_i(\bar{x},\bar{y})$, and every $C_i(\bar{x},\bar{y})$ is computed on at least one branch. When the signature contains a unary function symbol of the weight 0, the transformation will preserve a weaker form of equivalence: some solutions will be lost, but solvability will be preserved.

In our proof, we will reduce solvability of Knuth-Bendix ordering constraints to the problem of solvability of systems of linear Diophantine inequalities on the weights of variables. Condition 1 of the definition of the Knuth-Bendix order $|t| > |s|$ has a simple translation into a linear Diophantine inequality, but conditions 2 and 3 do not. So we will split the Knuth-Bendix order in two partial orders: $\succ_w$ corresponding to condition 1 and $\succ_{lex}$ corresponding to conditions 2 and 3. Formally, we denote by $t \succ_w s$ the formula $|t| > |s|$ and by $t \succ_{lex} s$ the formula $|t| =_{\mathbb{N}} |s| \wedge t \succ s$. Obviously, $t_1 \succ t_2$ if and only if $t_1 \succ_{lex} t_2 \vee t_1 \succ_w t_2$. So in the sequel we will assume that $\succ$ is replaced by the new symbols $\succ_{lex}$ and $\succ_w$.

We use $x_1 \succ x_2 \succ \ldots \succ x_n$ to denote the formula $x_1 \succ x_2 \wedge x_2 \succ x_3 \wedge \ldots \wedge x_{n-1} \succ x_n$, and similar for other binary symbols in place of $\succ$.

A term t is called *flat* if t is either a variable or has the form $g(x_1,\ldots,x_m)$, where $g \in \Sigma$, $m \geq 0$, and $x_1,\ldots,x_m$ are variables. We call a constraint *chained* if (i) it has a form $t_1 \# t_2 \# \ldots \# t_n$, where each occurrence of $\#$ is $\succ_w$, $\succ_{lex}$ or $=_{\mathrm{TA}}$, (ii) each term t_i is flat, (iii) if some of the t_i's has the form $g(x_1,\ldots,x_n)$, then $x_1,\ldots,x_n$ are some of the t_j's. Denote by $\bot$ the logical constant "false".

Lemma 1. *Any constraint C is equivalent to a disjunction $C_1 \vee \ldots \vee C_k$ of chained constraints. Moreover, there exists a nondeterministic polynomial-time algorithm which, for a given C, computes on every branch either $\bot$ or some C_i; and every C_i is computed on at least one branch.* □

We will now introduce several special kinds of constraints which will be used in our proofs below, namely *arithmetical*, *triangle*,*simple* and *isolated*. A constraint is called *arithmetical* if it uses only arithmetical relations $=_{\mathbb{N}}$ and $>$, for example $|f(x)| > |a| + 3$. A constraint $y_1 =_{\mathrm{TA}} t_1 \wedge \ldots \wedge y_n =_{\mathrm{TA}} t_n$ is said to be in *triangle form* if (i) $y_1,\ldots,y_n$ are pairwise different variables, and (ii) for all $j \geq i$ the variable y_i does not occur in t_j. The variables $y_1,\ldots,y_n$ are said to be *dependent* in this constraint. A constraint is said to be *simple* if it has the form

$$x_{11} \succ_{lex} x_{12} \succ_{lex} \ldots \succ_{lex} x_{1n_1} \wedge \ldots \wedge x_{k1} \succ_{lex} x_{k2} \succ_{lex} \ldots \succ_{lex} x_{kn_k},$$

where $x_{11},\ldots,x_{kn_k}$ are pairwise different variables. A constraint is said to be in *isolated form* if either it is $\bot$ or it has the form $C_{arith} \wedge C_{triang} \wedge C_{simp}$, where

C_{arith} is an arithmetical constraint, C_{triang} is in triangle form, and C_{simp} is a simple constraint such that no variable of C_{simp} is dependent in C_{triang}.

Our decision procedure for Knuth-Bendix ordering constraints is designed as follows. By Lemma 1 we can transform any constraint into an equivalent disjunction of chained constraints. Our next step is to give a transformation of any chained constraint into an equivalent disjunction of constraints in isolated form. Then in Section 4 we show how to transform any constraint in isolated form into an equivalent disjunction of systems of linear Diophantine inequalities on the weights of variables. Then we can use the result that the decidability of systems of linear Diophantine inequalities is in NP.

Let us show how to transform any chained constraint into an equivalent disjunction of isolated forms. The transformation will work on the constraints of the form

$$C_{chain} \wedge C_{arith} \wedge C_{triang} \wedge C_{simp}, \tag{1}$$

such that (i) $C_{arith}, C_{triang}, C_{simp}$ are as in the definition of isolated form; (ii) C_{chain} is a chained constraint; (iii) each variable of C_{chain} neither occurs in C_{simp} nor is dependent in C_{triang}. We will call such constraints (1) *working*. Let us call the *size* of a chained constraint C the total number of occurrences of function symbols and variables in C. Likewise, the *essential size* of a working constraint is the size of its chained part C_{chain}.

At each transformation step we will replace working constraint (1) by a disjunction of working constraints but of smaller essential sizes. Evidently, when the essential size is 0, we obtain a constraint in isolated form.

Let us prove some lemmas about solutions to constraints of the form (1). Note that any chained constraint is of the form

$$\begin{array}{c} t_{11}\#t_{12}\#\ldots\#t_{1m_1} \\ \succ_w \\ \cdots \\ \succ_w \\ t_{k1}\#t_{k2}\#\ldots\#t_{km_k}, \end{array} \tag{2}$$

where each $\#$ is either $=_{\mathrm{TA}}$ or $\succ_{lex}$ and each t_{ij} is a flat term. We call a *row* in such a constraint any maximal subsequence $t_{i1}\#t_{i2}\#\ldots\#t_{im_i}$ in which $\succ_w$ does not occur. So constraint (2) contains k rows, the first one is $t_{11}\#t_{12}\#\ldots\#t_{1m_1}$ and the last one $t_{k1}\#t_{k2}\#\ldots\#t_{km_k}$. Note that for any solution to (2) all terms in a row have the same weight.

Lemma 2. *There exists a polynomial-time algorithm which transforms any chained constraint C into an equivalent chained constraint that is either $\bot$, or of the form (2) and has the following property. Suppose some term of the first row t_{1j} is a variable y. Then either*

1. *y has exactly one occurrence in C, namely t_{1j} itself; or*
2. *y has exactly two occurrences in C, both in the first row: some t_{1n} has the form $f(y)$ for $n < j$, and $w(f) = 0$; moreover in this case there exists at least one $\succ_{lex}$ between t_{1n} and t_{1j}.* □

It is not hard to argue that the transformation of Lemma 2 does not increase the size of the constraint.

We will now take a working constraint $C_{chain} \wedge C_{arith} \wedge C_{triang} \wedge C_{simp}$, whose chained part satisfies Lemma 2 and transform it into an equivalent disjunction of working constraints of smaller essential sizes. More precisely, these constraints will be equivalent when the signature contains no unary function symbol of the weight 0. When the signature contains such a symbol f, a weaker notion of equivalence will hold.

A term s is called an *f-variant* of a term t if s can be obtained from t by a sequence of operations of the following forms: replacement of a subterm $f(r)$ by r or replacement of a subterm r by $f(r)$. Evidently, f-variant is an equivalence relation. Two substitutions θ_1 and θ_2 are said to be f-variants if for every variable x the term $x\theta_1$ is an f-variant of $x\theta_2$. In the proof of several lemmas below we will replace a constraint $C(\bar{x})$ by a formula $A(\bar{x}, \bar{y})$ containing extra variables $\bar{y}$ and say that $C(\bar{x})$ and $A(\bar{x}, \bar{y})$ are *equivalent up to f*. By this we mean the following.

1. For every substitution θ_1 grounding for $\bar{x}$ such that $\mathrm{TA}^+(\Sigma) \models C(\bar{x})\theta_1$, there exists a substitution θ_2 grounding for $\bar{x}, \bar{y}$ such that $\mathrm{TA}^+(\Sigma) \models A(\bar{x}, \bar{y})\theta_2$, and the restriction of θ_2 to $\bar{x}$ is an f-variant of θ_1.
2. For every substitution θ_2 grounding for $\bar{x}, \bar{y}$ such that $\mathrm{TA}^+(\Sigma) \models A(\bar{x}, \bar{y})\theta_2$, there exists a substitution θ_1 such that $\mathrm{TA}^+(\Sigma) \models C(\bar{x})\theta_1$ and θ_1 is an f-variant of the restriction of θ_2 to $\bar{x}$.

Note that when the signature contains no unary function symbol of the weight 0, equivalence up to f is the same as ordinary equivalence.

Lemma 3. *Let $C = C_{chain} \wedge C_{arith} \wedge C_{triang} \wedge C_{simp}$ be a working constraint and θ_1 be a solution to C. Let θ_2 be an f-variant of θ_1 such that (i) θ_2 is a solution to C_{chain} and (ii) θ_2 coincides with θ_1 on all variables not occurring in C_{chain}. Then there exists an f-variant θ_3 of θ_2 such that (i) θ_3 is a solution to C and (ii) θ_3 coincides with θ_2 on all variables except for the dependent variables of C_{triang}.* □

This lemma will be used below in the following way. Instead of considering the set Θ_1 of all solutions to C_{chain} we can restrict ourselves to a subset Θ_2 of Θ_1 as soon as for every solution $\theta_1 \in \Theta_1$ there exists a solution $\theta_2 \in \Theta_2$ such that θ_2 is an f-variant of θ_1.

Let us call an *f-term* any term of the form $f(t)$. By the *f-height* of a term t we mean the number n such that $t = f^n(s)$ and s is not an f-term. Note that the f-terms are exactly the terms of a positive f-height. We call the *f-distance* between two terms s and t the difference between the f-height of s and f-height of t. For example, the f-distance between the terms $f(a)$ and $f(f(g(a, b))$ is -1.

Let us now prove a lemma that restricts f-height of solutions.

Lemma 4. *Let C_{chain} be a chained constraint of the form $p_l \# p_{l-1} \# \ldots \# p_1 \succ_w \ldots$, where each $\#$ is either $=_{\mathrm{TA}}$ or $\succ_{lex}$. Further, let C_{chain} satisfy the conditions of Lemma 2 and θ be a solution to C_{chain}. Then there exists an f-variant θ' of*

θ such that (i) θ' is a solution to C_{chain} and (ii) for every $k \in \{1, \ldots, l\}$, the f-height of $p_k\theta'$ is at most k.

Proof. Let us first prove the following statement

(3) The row $p_l \# p_{l-1} \# \ldots \# p_1$ has a solution θ_1, such that (i) θ_1 is an f-variant of θ, (ii) for every $1 < k \leq l$ the f-distance between $p_k\theta_1$ and $p_{k-1}\theta_1$ is at most 1.

Suppose that for some k the f-distance between $p_k\theta$ and $p_{k-1}\theta$ is $d > 1$. Evidently, to prove (3) it is enough to show the following.

(4) There exists a solution θ_2 such that (i) θ_2 is an f-variant of θ, (ii) the f-distance between $p_k\theta_2$ and $p_{k-1}\theta_2$ is $d-1$, and (iii) for every $k' \neq k$ the f-distance between $p_{k'}\theta_2$ and $p_{k'-1}\theta_2$ coincides with the f-distance between $p_{k'}\theta$ and $p_{k'-1}\theta$.

Let us show (4), and hence (3). Since θ is a solution to the row, then for every $k''' \geq k$ the f-distance between any $p_{k'''}\theta$ and $p_k\theta$ is nonnegative. Likewise, for every $k'' < k-1$ the f-distance between any $p_{k-1}\theta$ and $p_{k''}\theta$ is nonnegative. Therefore, for all $k''' \geq k > k''$, the f-distance between $p_{k'''}\theta$ and $p_{k''}\theta$ is $\geq d$, and hence is at least 2. Let us prove the following.

(5) Every variable x occurring in the sequence $p_l \# p_{l-1} \# \ldots \# p_k$ does not occur in $p_{k-1} \# \ldots \# p_1$.

Let x occur in both $p_l \# p_{l-1} \# \ldots \# p_k$ and $p_{k-1} \# \ldots \# p_1$. Since the constraint satisfies Lemma 2, then $p_i = f(x)$ and $p_j = x$. Then the f-distance between $p_i\theta$ and $p_j\theta$ is 1, but by our assumption it is at least 2, so we obtain a contradiction. Hence (5) is proved.

Now note the following.

(6) If for some $k''' \geq k$ a variable x occurs in $p_{k'''}$, then $x\theta$ is an f-term.

Suppose, by contradiction, that $x\theta$ is not an f-term. Note that $p_{k'''}$ has a positive f-height, so $p_{k'''}$ is either x of $f(x)$. But we proved before that the f-distance between $p_{k'''}$ and p_{k-1} is at least 2, so x must be an f-term.

Now, to satisfy (4), define the substitution θ_2 as follows:

$$\theta_2(x) = \begin{cases} \theta(x), & \text{if } x \text{ does not occur in } p_l, \ldots, p_k, \\ t, & \text{if } x \text{ occurs in } p_l, \ldots, p_k \text{ and } \theta(x) = f(t). \end{cases}$$

By (5) and (6), θ_2 is defined correctly. We claim that θ_2 satisfies (4). The properties (i)-(iii) of (4) are straightforward by our construction, it only remains to prove that θ_2 is a solution to the row, i.e. for every k' we have $p_{k'}\theta_2 \# p_{k'-1}\theta_2$. Well, for $k' > k$ we have $p_{k'}\theta = f(p_{k'}\theta_2)$ and $p_{k'-1}\theta = f(p_{k'-1}\theta_2)$, and for $k' < k$ we have $p_{k'}\theta = p_{k'}\theta_2$ and $p_{k'-1}\theta = p_{k'-1}\theta_2$, in both cases $p_{k'}\theta_2 \# p_{k'-1}\theta_2$ follows from $p_{k'}\theta \# p_{k'-1}\theta$. The only difficult case is $k = k'$.

Assume $k = k'$. Since the f-distance between $p_k\theta$ and $p_{k-1}\theta$ is $d > 1$, we have $p_k\theta \neq p_{k-1}\theta$, and hence $p_k \# p_{k-1}$ must be $p_k \succ_{lex} p_{k-1}$. Since θ is a solution to $p_k \succ_{lex} p_{k-1}$ and since θ_2 is an f-variant of θ, the weights of $p_k\theta_2$ and $p_{k-1}\theta_2$ coincide. But then $p_k\theta_2 \succ_{lex} p_{k-1}\theta_2$ follows from the fact that the f-distance between $p_k\theta_2$ and $p_{k-1}\theta_2$ is $d - 1 \geq 1$.

Now the proof of (4), and hence of (3), is completed. In the same way as (3), we can also prove

(7) The constraint C_{chain} has a solution θ' such that (i) θ' is an f-variant of θ, (ii) for every $1 < k \leq l$ the f-distance between $p_k\theta_1$ and $p_{k-1}\theta'$ is at most 1. (iii) the f-height of $p_1\theta'$ is at most 1; (iv) θ' and θ coincide on all variables occurring in the rows below the first one.

It is not hard to derive Lemma 4 from (7).

Lemma 5. *Let $C = C_{chain} \wedge C_{arith} \wedge C_{triang} \wedge C_{simp}$ be a working constraint in which C_{chain} is nonempty. There exists a nondeterministic polynomial-time algorithm which transforms C into a disjunction of working constraints having C_{chain} of smaller sizes and equivalent to C up to f.*

Proof. The proof is rather complex, so we will give a plan of it. The proof is presented as a series of transformations on the first row of C. These transformations may result in new constraints added to C_{arith}, C_{triang}, and C_{simp}. First, we will get rid of equations $s =_{\mathrm{TA}} t$ in the first row, by introducing *quasi-flat* terms, i.e. terms $f^k(t)$, where t is flat. If the first row contained no function symbols, then we simply eliminate the first row, thus decreasing the size of the chained part. If there were function symbols in the first row, we continue as follows.

We "guess" the values of some variables x of the first row, i.e. replace them by some quasi-flat term $f^m(g(\bar{y}))$, where $\bar{y}$ is a sequence of new variables. After these steps, the size of the first row can, in general, increase. Then we show how to replace the first row by new constraints involving only variables occurring in the row, but not function symbols. Finally we prove that the number of variables from the new constraints that remain in the chained part is not greater than the original number of variables in the first row, and therefore the size of the chained part decreases.

Consider the first row of C_{chain}. Let this row be $p_l \# p_{l-1} \# \ldots \# p_1$. Then C_{chain} has the form $p_l \# p_{l-1} \# \ldots \# p_1 \succ_w t_1 \# \ldots \# t_n$. If $l = 1$, i.e., the first row consists of one term, we can remove this row and add $|p_1| > |t_1|$ to C_{arith} obtaining an equivalent constraint with smaller essential size. So we assume that the first row contains at least two terms.

As before, we assume that f is a unary function symbol of the weight 0. By Lemma 4, if some p_i is either a variable x or a term $f(x)$, it is enough to search for solutions θ such that the height of $x\theta$ is at most l.

A term is called *quasi-flat* if it has the form $f^k(t)$ where t is flat. We will now get rid of equalities in the first row, but by introducing quasi-flat terms instead of the flat ones. When we use notation $f^k(t)$ below, we assume $k \geq 0$, and $f^0(t)$ will stand for t. Let us first get rid of equalities of the form $f^k(x) =_{\mathrm{TA}} f^m(y)$ and then of all other equalities.

If the first row contains an equality $f^k(x) =_{\mathrm{TA}} f^m(y)$, we do the following. If x coincides with y and $k \neq m$, then the constraint is unsatisfiable. If x coincides with y and $k = m$, we replace $f^k(x) =_{\mathrm{TA}} f^m(y)$ by $f^k(x)$. Assume now that x is different from y. Without loss of generality assume $k \geq m$. We add $y =_{\mathrm{TA}} f^{k-m}(x)$ to C_{triang}, and replace all other occurrences of y in C_{chain} (if any) by $f^{k-m}(x)$. Note that other occurrences of y in C_{chain} can only be in the first row, and only in the terms $f^r(y)$.

After all these transformations we can assume that equalities $f^k(x) =_{\mathrm{TA}} f^m(y)$ do not occur in the first row.

If the first row contains an equality $x =_{\mathrm{TA}} t$ between a variable and a term, we replace this equality by t, replace all occurrences of x by t in the first row, and add $x =_{\mathrm{TA}} t$ to C_{triang} obtaining an equivalent working constraint.

If the first row contains an equality $g(x_1, \ldots, x_m) =_{\mathrm{TA}} h(t_1, \ldots, t_n)$ where g and h are different function symbols, the constraint is unsatisfiable.

If the first row contains an equality $g(x_1, \ldots, x_n) =_{\mathrm{TA}} g(y_1, \ldots, y_n)$ we do the following. If the term $g(x_1, \ldots, x_n)$ coincides with $g(y_1, \ldots, y_n)$, replace this equality by $g(x_1, \ldots, x_n)$. Otherwise, find the smallest number i such that x_i is different from y_i and (i) add $y_i =_{\mathrm{TA}} x_i$ to C_{triang}; (ii) replace all occurrences of y_i in C_{chain} by x_i.

So we can now assume that the first row contains no equalities and hence it has the form $q_n \succ_{lex} q_{n-1} \succ_{lex} \ldots \succ_{lex} q_1$, where all of the terms q_i are quasi-flat.

If all of the q_i are variables, we can move $q_n \succ_{lex} q_{n-1} \succ_{lex} \ldots \succ_{lex} q_1$ to C_{simp} and add $|q_1| > |t_1|$ to C_{arith} obtaining an equivalent working constraint of smaller essential size. Hence, we can assume that at least one of the q_i is a nonvariable term.

Take any term q_k in the first row such that q_k is either a variable x or a term $f^r(x)$. Note that other occurrences of x in C_{chain} can only be in the first row, and only in the terms of the form $f^k(x)$.

Consider the formula G defined as

$$\bigvee_{g \in \Sigma - \{f\}} \bigvee_{m=0 \ldots l} x =_{\mathrm{TA}} f^m(g(\bar{y})). \tag{8}$$

where $\bar{y}$ is a sequence of pairwise different new variables. Since we proved that it is enough to restrict ourselves to solutions θ for which the height of $x\theta$ is at most l, the formulas C and $C \wedge G$ are equivalent up to f.

Using the distributivity laws, $C \wedge G$ can be turned into an equivalent disjunction of formulas $x =_{\mathrm{TA}} f^m(g(\bar{y})) \wedge C$. For every such formula, replace x by $f^m(g(\bar{y}))$ in the first row, and add $x =_{\mathrm{TA}} f^m(g(\bar{y}))$ to the triangle part. We do this transformation for all terms in the first row of the form $f^k(z)$, where $k \geq 0$ and z is a variable.

Let us show how to replace constraints of the first row with equivalent constraints consisting of constraints on variables and arithmetical constraints. Consider the pair q_n, q_{n-1}. Now $q_n = f^k(g(x_1, \ldots, x_u))$ and $q_{n-1} = f^m(h(y_1, \ldots, y_v))$ for some variables $x_1, \ldots, x_u, y_1, \ldots, y_v$ and function symbols

$g, h \in \Sigma - \{f\}$. Then $q_n \succ_{lex} q_{n-1}$ is $f^k(g(x_1, \ldots, x_u)) \succ_{lex} f^m(h(y_1, \ldots, y_v))$. If $k < m$ or ($k = m$ and $h \gg g$), then $f^k(g(x_1, \ldots, x_u)) \succ_{lex} f^m(h(y_1, \ldots, y_v))$ is equivalent to $\bot$. If $k > m$ or ($k = m$ and $g \gg h$), then $f^k(g(x_1, \ldots, x_u)) \succ_{lex} f^m(h(y_1, \ldots, y_v))$ is equivalent to the arithmetical constraint $|g(x_1, \ldots, x_u)| =_{\mathbb{N}} |h(y_1, \ldots, y_v)|$ which can be added to C_{arith}. If $k = m$ and $g = h$ (and hence $u = v$), then

$$f^k(g(x_1, \ldots, x_u)) \succ_{lex} f^m(h(y_1, \ldots, y_v)) \leftrightarrow |g(x_1, \ldots, x_u)| =_{\mathbb{N}} |h(y_1, \ldots, y_v)| \wedge \bigvee_{i=1 \ldots u} (x_1 =_{\mathrm{TA}} y_1 \wedge \ldots \wedge x_{i-1} =_{\mathrm{TA}} y_{i-1} \ \wedge x_i \succ y_i).$$

We can now do the following. Add $|g(x_1, \ldots, x_u)| =_{\mathbb{N}} |h(y_1, \ldots, y_v)|$ to C_{arith} and replace $q_n \succ_{lex} q_{n-1}$ with the equivalent disjunction

$$\bigvee_{i=1 \ldots u} (x_1 =_{\mathrm{TA}} y_1 \wedge \ldots \wedge x_{i-1} =_{\mathrm{TA}} y_{i-1} \ \wedge x_i \succ y_i).$$

Then using the distributivity laws turn this formula into the equivalent disjunction of constraints of the form $C \wedge x_1 =_{\mathrm{TA}} y_1 \wedge \ldots \wedge x_{i-1} =_{\mathrm{TA}} y_{i-1} \wedge x_i \succ y_i$ for all $i = 1 \ldots u$. For each of these constraints, we can move, as before, the equalities $x =_{\mathrm{TA}} y$ one by one to the triangle part C_{triang}, and make $C_{chain} \wedge x_i \succ y_i$ into a disjunction of chained constraints as in Lemma 1.

Let us analyze what we have achieved. After these transformations, in each member of the obtained disjunction the first row is removed from the chained part C_{chain} of C. Since the row contained at least one function symbol, each member of the disjunction will contain at least one occurrence of a function symbol less than the original constraint. This is enough to prove termination of our algorithm, but not enough to present it as a nondeterministic polynomial-time algorithm. The problem is that, when p_n is a variable x or a term $f(x)$, one occurrence of x in p_n can be replaced by one or more constraints of the form $x_i \succ y_i$, where x_i and y_i are new variables. To be able to show that the essential sizes of each of the resulting constraints is strictly less than the essential size of the original constraint, we have to modify our algorithm slightly.

The modification will guarantee that the number of new variables introduced in the chained part of the constraint is not greater than the number of variables eliminated from the first row. We will achieve this by moving some constraints to the simple part C_{simp}.

The new variables only appear when we replace a variable in the first row by a term $f^k(h(u_1, \ldots, u_m))$ or by $f^k(h(v_1, \ldots, v_m))$ obtaining a constraint $f^k(h(u_1, \ldots, u_m)) \succ_{lex} f^k(h(v_1, \ldots, v_m))$, which is then replaced by

$$u_1 =_{\mathrm{TA}} v_1 \wedge \ldots \wedge u_{i-1} =_{\mathrm{TA}} v_{i-1} \ \wedge u_i \succ v_i. \tag{9}$$

Let us call a variable u_i (respectively, v_i) *new* if $f^k(h(u_1, \ldots, u_m))$ appeared in the first row when we replaced a variable by a nonvariable term containing h using formula (8). In other words, new variables are those that did not occur in the first row before our transformation, but appeared in the first row during the

transformation. All other variables are called *old*. After the transformation we obtain a conjunction E of constraints of the form $x_i =_{\mathrm{TA}} x_j$ or $x_i \succ x_j$, where x_i, x_j can be either new or old. Without loss of generality we can assume that this conjunction of constraints does not contain chains of the form $x_1 \# \ldots \# x_n \# x_1$ where $n \geq 2$ and at least one of the $\#$'s is $\succ$. Indeed, if E contains such a chain, then it is unsatisfiable.

We will now show that the number of new variables can be restricted by moving constraints on these variables into the triangle or simple part. Among the new variables, let us distinguish the following three kinds of variables. A new variable x is called *blue in* E if E contains a chain $x =_{\mathrm{TA}} x_1 =_{\mathrm{TA}} \ldots =_{\mathrm{TA}} x_n$, where x_n is an old variable. Evidently, a blue variable x causes no harm since it can be replaced by an old variable x_n. Let us denote by $\prec$ the inverse relation to $\succ$. A new variable x is called *red in* E if it is not blue in E and E contains a chain $x \# x_1 \# \ldots \# x_n$, where x_n is an old variable, and all of the $\#$'s are either $=_{\mathrm{TA}}$, or $\succ$, or $\prec$. Red variables are troublesome, since there is no obvious way to get rid of them. However, we will show that the number of red variables is not greater than the number of replaced variables (such as the variable x in (8)). Finally, all variables that are neither blue nor red in E are called *green* in E.

Getting rid of green variables. We will now show that the green variables can be moved to the simple part of the constraint C_{simp}. To this end, note an obvious property: if E contains a constraint $x \# y$ and x is green, then y is green too. We can now do the following with the green variables. As in Lemma 1, we can turn all the green variables into a disjunction of chained constraints of the form $v_1 \# \ldots \# v_n$, where $\#$ are $=_{\mathrm{TA}}$, $\succ_w$, or $\succ_{lex}$, and use the distributivity laws to obtain chained constraints $v_1 \# \ldots \# v_n$. Let us call this constraint a *green chain*. Then, if there is any equality $v_i =_{\mathrm{TA}} v_{i+1}$ in the green chain, we add this equality to C_{triang} and replace this equality by v_{i+1} in the chain. Further, if the chain has the form $v_1 \succ_{lex} \ldots \succ_{lex} v_k \succ_w v_{k+1} \# \ldots \# v_n$, we add $v_1 \succ_{lex} \ldots \succ_{lex} v_k$ to C_{simp} and $|v_k| > |v_{k+1}|$ to C_{arith}, and replace the green chain by $v_{k+1} \# \ldots \# v_n$. We do this transformation until the green chain becomes of the form $v_1 \succ_{lex} \ldots \succ_{lex} v_k$. After this, the green chain can be removed from E and added to C_{simp}. Evidently, this transformation can be presented as a nondeterministic polynomial-time algorithm.

Getting rid of blue variables. If E contains a blue variable x, then it also contains a chain of constraints $x =_{\mathrm{TA}} x_1 =_{\mathrm{TA}} \ldots =_{\mathrm{TA}} x_n$, where x_n is an old variable. We replace x by x_n in C and add $x =_{\mathrm{TA}} x_n$ to the triangle part C_{triang}.

Red variables. Let us show the following: in every term $f^k(h(u_1, \ldots, u_m))$ in the first row at most one variable among $u_1, \ldots, u_m$ is red. It is not hard to argue that it is sufficient to prove a stronger statement: if for some i the variable u_i is red or blue, then all variables $u_1, \ldots, u_{i-1}$ are blue. So suppose u_i is red and $u_i \# y_n \# \ldots \# y_1$ is a shortest chain in E such that y_1 is old. We prove that the variables $u_1, \ldots, u_{i-1}$ are blue, by induction on n. When $n = 1$, E contains either $u_i \succ y_1$ or $y_1 \succ u_i$, where y_1 is old. Without loss of generality assume that E contains $u_i \succ y_1$. Then (cf. (9)) this equation appeared in E when we replaced $f^k(h(u_1, \ldots, u_m)) \succ_{lex} f^k(h(v_1, \ldots, v_m))$ by $u_1 =_{\mathrm{TA}} v_1 \wedge \ldots \wedge$

$u_{i-1} =_{\mathrm{TA}} v_{i-1} \wedge u_i \succ v_i$ and $y_1 = v_i$. But then E also contains the equations $u_1 =_{\mathrm{TA}} v_1, \ldots, u_{i-1} =_{\mathrm{TA}} v_{i-1}$, where the variables $v_1, \ldots, v_{i-1}$ are old, and so the variables $u_1, \ldots, u_{i-1}$ are blue. In the same way we can prove that if u_i is blue then $u_1, \ldots, u_{i-1}$ are blue. The proof for $n > 1$ is similar, but we use the fact that $v_1, \ldots, v_{i-1}$ are blue rather than old.

To complete the transformation, we add all constraints on red variables to C_{chain} and make C_{chain} into a disjunction of chained constraint as in Lemma 1.

When we completed the transformation on the first row, the row disappears from the chained part C_{chain} of C. If the first row contained no function symbols, the size of C_{chain} will become smaller, since several variables will be removed from it. If C_{chain} contained at least one function symbol, that after the transformation the number of occurrences of function symbols in C_{chain} will decrease. Some red variables will be introduced, but we proved that their number is not greater than the number of variables eliminated from the first row. Therefore, the size of C_{chain} strictly decreases after the transformation.

Again, it is not hard to argue that the transformation can be presented as a nondeterministic polynomial-time algorithm computing all members of the resulting disjunction of constraints.

Lemmas 1 and 5 imply the following:

Lemma 6. *Let C be a constraint. Then there exists a disjunction $C_1 \vee \ldots \vee C_n$ of constraints in isolated form equivalent to C up to f. Moreover, members of such a disjunction can be found by a nondeterministic polynomial-time algorithm.*

4 From Constraints in Isolated Form to Systems of Linear Diophantine Inequalities

Let C be a constraint in isolated form $C_{simp} \wedge C_{arith} \wedge C_{triang}$. Our decision algorithm will be based on a transformation of the simple constraint C_{simp} into an equivalent disjunction D of arithmetical constraints. Then we can check the satisfiability of the resulting formula $D \wedge C_{arith}$ by using an algorithm for solving systems of linear Diophantine inequalities on the weights of variables.

To transform C_{simp} into an arithmetical formula, observe the following. The constraint C_{simp} is a conjunction of the constraints of the form $x_1 \succ_{lex} \ldots \succ_{lex} x_N$ having no common variables. To solve such a constraint we have to ensure that at least N different terms of the same weight as x_1 exist.

Lemma 7. *There exists a polynomial time of N algorithm, which constructs an existential formula $at\_least_N(x)$ of Presburger arithmetic valid on a natural number x if and only if there exists at least N different terms of the weight x.* □

5 Main Result

Theorem 1. *Knuth-Bendix ordering constraint solving is NP-complete.*

Proof. Take a constraint. By Lemma 5 it can be effectively transformed into an equivalent disjunction of isolated forms, so it remains to show how to check satisfiability of constraints in isolated form. Suppose that C is in isolated form

$$C_{arith} \wedge C_{triang} \wedge C_{simp}. \tag{10}$$

Let C_{simp} contain a chain $x_1 \succ_{lex} \ldots \succ_{lex} x_N$ such that $x_1, \ldots, x_N$ does not occur in the rest of C_{simp}. Denote by C'_{simp} the constraint obtained from C_{simp} by removing this chain. It is not hard to argue that C is equivalent to the constraint

$$C_{arith} \wedge C_{triang} \wedge C'_{simp} \wedge \bigwedge_{i=2\ldots N} (|x_i| =_{\mathbb{N}} |x_1|) \wedge at\_least_N(|x_1|).$$

In this way we can replace C_{simp} by an arithmetical constraint, so we assume that C_{simp} is empty. Let C_{triang} have the form $y_1 =_{\mathrm{TA}} t_1 \wedge \ldots \wedge y_n =_{\mathrm{TA}} t_n$. Let Z be the set of all variables occurring in $C_{arith} \wedge C_{triang}$. It is not hard to argue that $C_{arith} \wedge C_{triang}$ is satisfiable if and only if the following constraint is satisfiable:

$$C_{arith} \wedge |y_1| =_{\mathbb{N}} |t_1| \wedge \ldots \wedge |y_n| =_{\mathbb{N}} |t_n| \wedge \textstyle\bigwedge_{z \in Z} at\_least_1(|z|).$$

So we reduced the decidability of the existential theory of term algebras with a Knuth-Bendix order to the problem of solvability of systems of linear Diophantine inequalities. Our proof can be represented as a nondeterministic polynomial-time algorithm.

Let us refer to [9] where it is shown that the problem is NP-hard by reduction of the solvability of systems of linear Diophantine equations to the solvability of the Knuth-Bendix ordering constraints.

This result and Theorem 1 implies the main result of this paper.

Theorem 2. *The existential first-order theory of any term algebra with the Knuth-Bendix order is NP-complete.*

References

1. F. Baader and T. Nipkow. *Term Rewriting and and All That.* Cambridge University press, Cambridge, 1998.
2. H. Comon. Solving symbolic ordering constraints. *International Journal of Foundations of Computer Science*, 1(4):387–411, 1990.
3. N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.
4. Th. Hillenbrand, A. Buch, R. Vogt, and B. Löchner. Waldmeister: High-performance equational deduction. *Journal of Automated Reasoning*, 18(2):265–270, 1997.
5. W. Hodges. *Model theory.* Cambridge University Press, 1993.

6. J.-P. Jouannaud and M. Okada. Satisfiability of systems of ordinal notations with the subterm property is decidable. In J.L. Albert, B. Monien, and M. Rodríguez-Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium, ICALP'91*, volume 510 of *Lecture Notes in Computer Science*, pages 455–468, Madrid, Spain, 1991. Springer Verlag.
7. H. Kirchner. On the use of constraints in automated deduction. In A. Podelski, editor, *Constraint Programming: Basics and Tools*, volume 910 of *Lecture Notes in Computer Science*, pages 128–146. Springer Verlag, 1995.
8. D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.
9. K. Korovin and A. Voronkov. A decision procedure for the existential theory of term algebras with the Knuth-Bendix ordering. In *Proc. 15th Annual IEEE Symp. on Logic in Computer Science*, pages 291–302, Santa Barbara, California, June 2000.
10. K. Korovin and A. Voronkov. Knuth-Bendix constraint solving is NP-complete. Preprint CSPP-8, Department of Computer Science, University of Manchester, February 2000.
11. P. Narendran, M. Rusinowitch, and R. Verma. RPO constraint solving is in NP. In G. Gottlob, E. Grandjean, and K. Seyr, editors, *Computer Science Logic, 12th International Workshop, CSL'98*, volume 1584 of *Lecture Notes in Computer Science*, pages 385–398. Springer Verlag, 1999.
12. R. Nieuwenhuis. Simple LPO constraint solving methods. *Information Processing Letters*, 47:65–69, 1993.
13. R. Nieuwenhuis. Rewrite-based deduction and symbolic constraints. In H. Ganzinger, editor, *Automated Deduction—CADE-16. 16th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, pages 302–313, Trento, Italy, July 1999.
14. A. Riazanov and A. Voronkov. Vampire. In H. Ganzinger, editor, *Automated Deduction—CADE-16. 16th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, pages 292–296, Trento, Italy, July 1999.
15. S. Schulz. System abstract: E 0.3. In H. Ganzinger, editor, *Automated Deduction—CADE-16. 16th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, pages 297–301, Trento, Italy, July 1999.
16. C. Weidenbach, B. Afshordel, U. Brahm, C. Cohrs, T. Engel, E. Keen, C. Theobalt, and D. Topic. System description: SPASS version 1.0.0. In H. Ganzinger, editor, *Automated Deduction—CADE-16. 16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 378–382, Trento, Italy, July 1999.

Amalgamation in CASL via Enriched Signatures

Lutz Schröder[1], Till Mossakowski[1], and Andrzej Tarlecki[2,3]

[1] BISS, Department of Computer Science, Bremen University
[2] Institute of Informatics, Warsaw University
[3] Institute of Computer Science, Polish Academy of Sciences

Abstract. We construct a representation of the institution of the algebraic specification language CASL in an institution called enriched CASL. Enriched CASL satisfies the amalgamation property, which fails in the CASL institution, as well as its converse. Thus, the previously suggested institution-independent semantics of architectural specifications is actually applicable to CASL. Moreover, a variety of results for institutions with amalgamation, such as computation of normal forms and theorem proving for structured specifications, can now be used for CASL.

Introduction

The use of formal methods within the development process of software systems is important especially for complex or safety-critical systems. Here, 'formal' implies that the specification is based on a logical system, with a rigorously defined syntax and semantics. It has been recognized that structuring operations for the specification of software systems can be studied largely independently of the underlying logical system; the most prominent formalization of this concept is the notion of *institution* [14]. Several institution independent languages for structuring specifications, among them ASL+ [2,24,25], the module algebra in [11], ACT TWO [12,13], and development graphs [21], have been investigated. The recently developed language CASL [9,10,22] has an institution independent semantics of both structured and architectural specifications (where the latter have the purpose of structuring *implementations* rather than specifications). An institution independent proof system for ASL-like specifications (parametrized over a proof system for the respective institution) is presented in [6].

For each of these institution independent specification languages, central results require the underlying institution to have the so-called amalgamation property (also referred to as exactness). Roughly speaking, this property states that models of given signatures can be combined to yield a uniquely determined model of a compound signature, provided that the original models are mutually compatible. The amalgamation property allows the computation of normal forms for specifications [4,6]; it is a prerequisite for good behaviour w.r.t. parametrization [13] and conservative extensions [11]. The combination of implementations in the semantics of architectural specifications crucially depends on amalgamation [27]. The proof system for development graphs with hiding [21], which allow a

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 993–1004, 2001.

management of change for structured specifications, is sound only for institutions with amalgamation. Last but not least, a Z-like state based language over an arbitrary institution with amalgamation has been developed in [3].

Many standard logical systems (like multisorted equational and first-order logic with the respective standard notions of model) admit amalgamation, so quite often this property is taken for granted in work on specification formalisms (cf. e.g. [25]). However, the expected amalgamation property fails in the setting of order-sorted algebra (when subsort relations are interpreted as arbitrary injections[1]), in particular in the CASL institution. Generally, the amalgamation property fails to hold if there are components in the models that are not named in the signatures, e.g. the implicit universe in unsorted first-order logic (which destroys amalgamation for disjoint unions of signatures), the implicit set of kinds in LF [15], the implicit set of worlds in temporal or modal logics, or the implicit subsort injections in the CASL logic.

The solution proposed here is to represent the CASL institution in an institution (called enriched CASL) that does have amalgamation. In this way, most of the results cited above can be applied to CASL by forming compound (colimit) signatures in the enriched signature category; the details of how this affects the actual work with CASL specifications are briefly discussed in Section 5. The institution independent form of this technique is used in the semantics of CASL architectural specifications as outlined in [27].

We refer to [1,19] for categorical terminology left unexplained here.

1 Institutions

The considerations ahead deal with aspects of the notion of *institution* [14]. An institution I consists of a category **Sign** of *signatures*, a *model functor*

$$\mathbf{Mod} : \mathbf{Sign}^{op} \to \mathbf{CAT},$$

where **CAT** denotes the quasicategory of categories and functors [1], and further components which formalize sentences and satisfaction. In this context, we need only the category of signatures and the model functor.

A cocone for a diagram in **Sign** is called *amalgamable* if it is mapped to a limit under **Mod**. I (or **Mod**) has the *(finite) amalgamation property* if (finite) colimit cocones in **Sign** are amalgamable, i.e. if **Mod** preserves (finite) limits. If, conversely, **Mod** reflects limits, I (or **Mod**) is called *definitionally complete*. (Possibly more appropriate notions such as 2-limits or bilimits [5] are out of the scope of this paper). Definitional completeness, which makes the colimit property a necessary condition for amalgamability of cocones, is required, e.g., to ensure 'almost completeness' for the static architectural semantics of [27]. It is easily seen that, under cocompleteness of the signature category, a model functor that has the amalgamation property is definitionally complete iff it reflects

[1] This overcomes the drawbacks of the subsorts-as-inclusions approach, namely that satisfaction is not closed under isomorphism, unless extra conditions like local filtration are assumed, which, however, behave bad w.r.t. colimits.

isomorphisms. Informally speaking, definitional completeness means that identifying symbols, adding a new symbol to a signature (without constraining it by axioms), or altering 'properties' of a symbol always modifies the model class.

2 Standard CASL Signatures

The specification language CASL (*Common Algebraic Specification Language*) has been designed by CoFI, the international *Common Framework Initiative for Algebraic Specification and Development* [9]. Its features include first-order logic, partial functions, subsorts, and structured and architectural specifications.

We sketch the definition of CASL signatures and their models; for further details see [7,10]. A *(standard)* CASL *signature* Σ consists of a preordered set S of sorts and sets of total and partial function symbols and predicate symbols. Function and predicate symbols are written $f : w \to s$ and $p : w$, respectively, where s is a sort and w is a string of sorts, thus determining their *name* and *profile*. Symbols with identical names are said to be in an *overloading* relation if their argument sorts have a common subsort and their result sorts (in the case of function symbols) have a common supersort. Partial function symbols may become total on subsorts, but not vice versa.

A *signature morphism* consists of an order-preserving map between the associated sort preorders and maps between the symbol sets that are compatible with symbol profiles, preserve totality (i.e. may map partial to total function symbols, but not vice versa), and preserve overloading. This defines the (cocomplete) signature category **CASLsign**.

A *model* of a CASL signature is an interpretation of the sorts by sets and of the sort preorder by *injective maps* between these sets such that composition and identities are preserved (in other words: a functor from the thin category associated to the sort preorder into the category of sets and injective maps), of the partial (total) function symbols by partial (total) functions between the sets specified by their profiles, and of the predicate symbols by relations. The interpretations of overloaded symbols are required to agree on common subsorts of the argument sorts via the corresponding subsort injections. *Model morphisms* are defined by the usual homomorphism condition (also w.r.t. the subsort embeddings) and preservation of predicate satisfaction.

Thus, we have a category $\mathbf{Mod}_{CASL}(\Sigma)$ of Σ-models; this assigment extends to a model functor $\mathbf{Mod}_{CASL} : \mathbf{CASLsign}^{op} \to \mathbf{CAT}$ in the standard way. It is well known that amalgamation fails for this model functor; the simplest example is the following:

Example 1. Let Σ be the signature with sorts s and t (and no operations), and let Σ_1 be the extension of Σ by the subsort relation $s < t$. Then the pushout

$$\begin{array}{ccc} \Sigma & \longrightarrow & \Sigma_1 \\ \downarrow & & \downarrow \\ \Sigma_1 & \longrightarrow & \Sigma_1 \end{array}$$

in **CASLsign** fails to be amalgamable (since two models of Σ_1, compatible w.r.t. the inclusion of Σ, may interpret the subsort embedding by different injections).

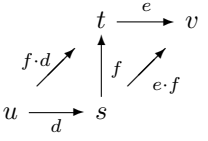
Incidentally, $\mathbf{Mod}_{CASL}$ also fails to be definitionally complete: e.g., the inclusion $\Sigma_1 \hookrightarrow \Sigma_2$, where Σ_1 contains sorts $s < t$ and an operation $f : s$ and Σ_2 contains an additional symbol $f : t$, induces an isomorphism between the two model categories.

3 Enriched Signatures

We now introduce a category of enriched signatures in which standard signatures can be represented via a suitable functor. Moreover, we equip this signature category with a model functor which has the amalgamation property and is definitionally complete, and which extends the original model functor up to a natural isomorphism; this enables us to treat amalgamability in the extended setting.

Example 1 suggests that the failure of amalgamation for standard signatures can be remedied by replacing the sort preorder by a sort category that admits more than one embedding between two sorts (similar category sorted algebras, although without a view on amalgamation, appear in [23]). This will be the main feature to distinguish enriched and standard signatures. We will certainly continue to require embeddings to be monomorphic; categories in which all morphisms are monomorphisms will be called *left cancellable*.

Moreover, there is an elegant way of handling the overloading of function and predicate symbols: introducing left and right actions of the sort category on the symbols. In the case of a unary function symbol $f : s \to t$ and sort embeddings $d : u \to s$, $e : t \to v$, these actions give rise to function symbols $f \cdot d : u \to t$ and $e \cdot f : s \to v$:



(The right action also applies to predicate symbols.) The appropriate behaviour of models w.r.t. overloading can then be ensured by requiring that the diagrams that arise from the actions as above are translated to commutative diagrams of maps (operations) in the models. This requirement replaces the rather cumbersome overloading axioms for models needed in the case of standard signatures; similarly, overloading preservation for signature morphisms now becomes a much more straightforward equivariance condition.

Thus, we arrive at the following

Definition 2. An *enriched (*CASL*) signature* Σ consists of

(i) a left cancellable *sort category* **S** with morphisms called *embeddings*;
(ii) a class F of *function symbols* and a class P of *predicate symbols*; symbols have profiles as in the case of standard signatures;

(iii) a unary *totality* predicate on F;
(iv) a *left action* of $\mathbf{S}$ on F which assigns to each function symbol $f : w \to s$ and each sort morphism $e : s \to t$ a function symbol $e \cdot f : w \to t$;
(v) a *right (multi)action* of $\mathbf{S}$ on F which assigns to each function symbol $f : s_1 \dots s_n \to t$ and each tuple of sort embeddings $d = (d_i : u_i \to s_i)_{i=1,\dots,n}$ a function symbol $f \cdot d : u_1 \dots u_n \to t$, and a similar right action on P.

These data are subject to the following axioms:

(i) The associative law and the identity law hold in the obvious sense (e.g., in the above notation, $id \cdot f = f$ and $(e \cdot f) \cdot d = e \cdot (f \cdot d)$).
(ii) For a sort embedding e and function symbols f, g with appropriate profiles, $e \cdot f = e \cdot g$ implies $f = g$.
(iii) Let f, e, and d be as above. If f is total, then $f \cdot d$ and $e \cdot f$ are total. Moreover, if $e \cdot f$ is total, then f is total.

A *signature morphism* σ between enriched signatures consists of a functor between the sort categories and a pair of maps between the classes of function and predicate symbols, respectively; all three components are denoted by σ. σ is required to be compatible with symbol profiles and to preserve totality. Moreover, σ is assumed to be *equivariant* w.r.t. the actions of the sort categories; i.e. if f, e, and d are as above, then

$$\sigma(e) \cdot \sigma(f) = \sigma(e \cdot f) \quad \text{and} \quad \sigma(f) \cdot \sigma(d) = \sigma(f \cdot d),$$

similarly for predicate symbols. An enriched signature is called *small* if its sort category and its symbol classes are small. Small enriched signatures and signature morphisms form a category **enrCASLsign**.

More precisely, one should say that **enrCASLsign** is a 2-category: 2-cells between signature morphisms are natural transformations between the functor parts that satisfy the obvious naturality condition w.r.t. symbols. In particular, one has a notion of equivalence of enriched signatures defined in the usual way via 'inverses up to isomorphism'. As in the case of categories, it is straightforward to show that such a definition of equivalence is equivalent to the following:

Definition 3. A signature morphism σ is called an *equivalence* if the functor part of σ is an equivalence, σ is, in the obvious sense, full and faithful on symbols, and whenever $\sigma(f)$ is a total function symbol, then so is f.

A crucial point is that the collection of all sets and partial maps can now be regarded as a signature. (Due to the rather different treatment of overloading, there is no obvious way to make this work for the signatures defined in [23]). More precisely: we have an enriched signature which has the category of sets and injective maps as sort category and n-ary partial functions and relations as function and predicate symbols, respectively, with the obvious totality predicate and assignment of profiles. The actions of the sort category are given by composition (in the case of the right action, by composition with cartesian products of maps or, for predicates, by taking preimages under such products). This enriched signature will be denoted by $\mathbf{Set_p}$.

This enables us to define models in the spirit of Lawvere [18]:

Definition 4. A model of an enriched signature Σ with sort category $\mathbf{S}$ is a signature morphism

$$\Sigma \to \mathbf{Set_p}.$$

A *morphism* between two such models σ, τ is a natural transformation μ between σ and τ, regarded as functors $\mathbf{S} \to \mathbf{Set}$, such that the usual homomorphism condition w.r.t. function symbols holds and satisfaction of predicates is preserved.

Thus, we have a *model category* of Σ, which we denote by $\mathbf{Mod}_{enr}(\Sigma)$. A signature morphism $\Sigma_1 \to \Sigma_2$ induces, via composition of signature morphisms, a functor $\mathbf{Mod}_{enr}(\Sigma_2) \to \mathbf{Mod}_{enr}(\Sigma_1)$, so that we have a model functor $\mathbf{Mod}_{enr} : \mathbf{enrCASLsign}^{op} \to \mathbf{CAT}$.

Now the representing functor $\mathsf{Enr} : \mathbf{CASLsign} \to \mathbf{enrCASLsign}$ acts on standard signatures by first forming a suitable completion of the symbol sets — closing symbol profiles under the sort preorder — to account for the actions of the sort embeddings, and then reinterpreting the data in the usual way (i.e. the sort preorder is interpreted as a thin category, and the actions of the sort category are defined using the mentioned closure of symbol profiles). Thanks to overloading preservation, this assignment on objects extends to a functor as required. Note that Enr is faithful, but is neither full nor injective on objects and, of course, does *not* preserve colimits.

It is easily verified that one indeed has a natural isomorphism

$$\mathbf{Mod}_{enr} \circ \mathsf{Enr}^{op} \to \mathbf{Mod}_{CASL}.$$

In particular, as indicated above,

Proposition 5. *A cocone in* **CASLsign** *is amalgamable w.r.t.* $\mathbf{Mod}_{CASL}$ *iff its image under* Enr *is amalgamable w.r.t.* $\mathbf{Mod}_{enr}$.

Finally,

Proposition 6. enrCASLsign *is cocomplete.*

4 Amalgamation and Definitional Completeness

In order to prove amalgamation and definitional completeness for enriched CASL, we introduce a further representation of enriched signatures in a suitable class of small categories which is easily seen to have the said properties; we then go on to show that the representation is sufficiently well-behaved to transfer these properties back to the setting of enriched signatures. In an intermediate step, we introduce a coding of enriched signatures as equational partial specifications in the sense of [8], thus essentially providing an extension of the existing embedding of the CASL logic into partial first-order logic with equality. Full proofs, mostly omitted here, can be found in the forthcoming [26].

Let **Lex** denote the category of small left exact (lex) categories (i.e. small categories with finite limits) and left exact (lex, finite limit preserving) functors. The *model category* $\mathbf{Mod}_{lex}(\mathbf{A})$ of a small lex category $\mathbf{A}$ is the category of lex

functors $\mathbf{A} \to \mathbf{Set}$ and natural transformations. By composition of lex functors, this assignment extends to a functor

$$\mathbf{Mod}_{lex} : \mathbf{Lex}^{op} \to \mathbf{CAT}.$$

$\mathbf{Mod}_{lex}$ trivially has the amalgamation property, i.e. preserves limits. Moreover, being a 2-functor, $\mathbf{Mod}_{lex}$ preserves equivalences. Conversely,

Theorem 7. $\mathbf{Mod}_{lex}$ *reflects equivalences.*

This is an immediate consequence of the fact that the finitely presentable objects in $\mathbf{Mod}_{lex}(\mathbf{A})$ are precisely the representable functors [20].

An (equational partial) *specification* $\mathcal{S} = (\Omega, \mathcal{A})$ is defined (slightly extending the definition in [8]) as follows:

Ω is a *signature* consisting of sorts, including a *terminal* sort T, and (partial) operation symbols with profiles as before, including a distinguished operation $\iota : \mathrm{T}$. This signature gives rise to a notion of term (with indicated profile) in the usual way. For convenience, tuples of terms are regarded as terms; so if $\alpha_i : w \to s_i$ are terms for $i = 1, \ldots, n$, then we write $\alpha = (\alpha_i) : w \to v$, where $v = s_1 \ldots s_n$. x_v denotes 'the' term $(x_1, \ldots, x_n)$, where x_i is a variable of sort s_i, $i = 1, \ldots, n$. If β is a term in the x_i, then $\beta\alpha$ denotes the term obtained by simultaneously substituting the α_i for the x_i.

$\mathcal{A}$ is a set of *axioms* that take the form of finite implications: an *existential equation* is a pair $\phi = (\alpha_1, \alpha_2)$ of terms (with identical profiles), written $\alpha_1 \stackrel{e}{=} \alpha_2$, and a *finite implication* is an implication of the form $\phi \Rightarrow \psi$, where ϕ and ψ are existential equations. By virtue of the tuple notation, the definition of existential equation covers finite conjunctions of such equations; we use notations like $\phi \wedge \psi$ with the obvious meaning. For $\phi = (\alpha_1, \alpha_2)$, $\phi\beta$ denotes $\alpha_1\beta \stackrel{e}{=} \alpha_2\beta \wedge \beta \stackrel{e}{=} \beta$. Equations $\alpha \stackrel{e}{=} \alpha$ are sometimes abbreviated as $\mathrm{def}\,\alpha$.

In Figure 1, we present the rules of a deduction system for existential equality associated to a specification $\mathcal{S} = (\Omega, \mathcal{A})$. This system is obviously sound w.r.t. the notion of model defined below; it will turn out to be complete as well. We write $\Phi \vdash \phi$ if an existential equation ϕ can be deduced from a set Φ of existential equations by means of these rules. If $\phi \vdash \psi$ for existential equations ϕ, ψ, we say that $\phi \Rightarrow \psi$ is a *theorem*.

$$\frac{s \text{ is a sort}}{x_s \stackrel{e}{=} x_s} \qquad \frac{}{x_{\mathrm{T}} \stackrel{e}{=} \iota} \qquad \frac{\alpha \stackrel{e}{=} \beta}{\beta \stackrel{e}{=} \alpha} \qquad \frac{\alpha \stackrel{e}{=} \beta,\ \beta \stackrel{e}{=} \gamma}{\alpha \stackrel{e}{=} \gamma} \qquad \frac{\phi, \psi}{\phi \wedge \psi} \qquad \frac{\phi \wedge \psi}{\phi} \qquad \frac{\phi \wedge \psi}{\psi}$$

$$\frac{\alpha \stackrel{e}{=} \alpha \quad \beta \text{ is a subterm of } \alpha}{\beta \stackrel{e}{=} \beta} \qquad \frac{\phi \Rightarrow \psi \in \mathcal{A} \quad \phi\alpha}{\psi\alpha} \qquad \frac{\alpha_1 \stackrel{e}{=} \alpha_2, \beta_1 \stackrel{e}{=} \beta_2 \quad \alpha_i\beta_i \text{ is well-formed, } i = 1, 2}{\alpha_1\beta_1 \stackrel{e}{=} \alpha_2\beta_2} \quad (*)$$

Fig. 1. Deduction rules for existential equality

A morphism between specifications is defined as a theory morphism in the usual sense, i.e. a signature morphism which preserves the terminal sort (and the distinguished operation) and transforms axioms into theorems. Thus, specifications form a category **epSpec**.

A *model* of the signature Ω is an algebra that interprets all symbols as partial operations in the usual way, with T interpreted as a fixed singleton set. An existential equation is satisfied in a model iff both terms are defined and equal for all interpretations of the variables; the satisfaction of a finite implication is defined in the obvious way. A model of $\mathcal{S} = (\Omega, \mathcal{A})$ is a model of Ω that satisfies all finite implications in $\mathcal{A}$; it is clear how this leads to a model functor $\mathbf{Mod}_{eps} : \mathbf{epSpec}^{op} \to \mathbf{CAT}$.

Using the deduction system of Figure 1, we can now construct a lex category $\mathsf{Th}_{eps}(\mathcal{S})$ from $\mathcal{S}$: the objects of $\mathsf{Th}_{eps}(\mathcal{S})$ are pairs (w, ϕ), where w is a string of sorts and ϕ is an existential equation in the variables of x_w. Morphisms $(w, \phi) \to (v, \psi)$ are terms $\alpha : w \to v$ such that $\phi \vdash \psi\alpha$, taken modulo existential equality deducible from ϕ. The identity on (w, ϕ) is represented by x_w. Composition is defined via simultaneous substitution of representing terms. This is a well-defined operation thanks to the congruence rule $(*)$ in Figure 1 and the following metatheorem, which also shows that the composite is indeed a morphism:

Proposition 8. *If ϕ and ψ are existential equations in variables x_i of sorts s_i and α is a tuple of terms α_i of sorts s_i, $i = 1, \ldots, n$, then $\phi \vdash \psi$ implies $\phi\alpha \vdash \psi\alpha$.*

As expected, 'concatenation' of objects defines finite products, and $x_w : (w, \phi \wedge \alpha_1 \overset{e}{=} \alpha_2) \to (w, \phi)$ is an equalizer of $\alpha_1, \alpha_2 : (w, \phi) \to (v, \psi)$ in $\mathsf{Th}_{eps}(\mathcal{S})$. Thus, $\mathsf{Th}_{eps}(\mathcal{S})$ is indeed a lex category. Similarly as in [8], one easily verifies that one has an equivalence of categories $\mathbf{Mod}_{lex}(\mathsf{Th}_{eps}(\mathcal{S})) \to \mathbf{Mod}_{eps}(\mathcal{S})$ (natural in $\mathcal{S}$) and concludes, using the fact that representable functors are models of $\mathsf{Th}_{eps}(\mathcal{S})$, that the deduction system of Figure 1 is complete.

It is straightforward to translate an enriched CASL signature Σ into an equational partial specification $\mathsf{Spec}(\Sigma) = (\Omega, \mathcal{A})$: the sorts in Ω are the sorts of Σ and a new terminal sort T; the symbols in Ω are the function and predicate symbols of Σ (where the range of all predicate symbols is T), the embedding symbols with the obvious unary profiles, and a new symbol ι : T. The axioms are given in Figure 2. It is easily verified that $\mathbf{Mod}_{eps} \circ \mathsf{Spec}^{op}$ is naturally isomorphic to $\mathbf{Mod}_{enr}$.

Remark 9. The translation functor Spec can be modified to retain predicates and include an elementhood predicate and partial projection functions for subsorts, thus providing an extension to enriched CASL of the existing embedding of the CASL logic into partial first-order logic with equality [7].

Putting the two translations together, we have a representation $\mathsf{Th}_{enr} := \mathsf{Th}_{eps} \circ \mathsf{Spec}$ of enriched signatures as lex categories. Moreover, we have a natural 'forgetful functor' $\mathsf{U} : \mathbf{Lex} \to \mathbf{enrCASLsign}$: given a lex category $\mathbf{A}$, the sorts of $\mathsf{U}(\mathbf{A})$ are the objects of $\mathbf{A}$, and the function symbols are the partial morphisms (spans containing a monomorphism, taken modulo isomorphy) with profiles determined by the finite product structure. Such a symbol is total iff the

$$\left.\begin{array}{l}\mathrm{def}(e \cdot f)(x_w) \Rightarrow (e \cdot f)(x_w) \stackrel{e}{=} e(f(x_w)) \\ \mathrm{def}\, e(f(x_w)) \Rightarrow (e \cdot f)(x_w) \stackrel{e}{=} e(f(x_w))\end{array}\right\} \text{ if } f : w \to s \text{ is a function symbol and } e : s \to t \text{ is an embedding;}$$

$$\left.\begin{array}{l}\mathrm{def}(f \cdot d)(x_v) \Rightarrow (f \cdot d)(x_v) \stackrel{e}{=} f(d(x_v)) \\ \mathrm{def}\, f(d(x_v)) \Rightarrow (f \cdot d)(x_v) \stackrel{e}{=} f(d(x_v))\end{array}\right\} \text{ if } f \text{ is a function or predicate symbol and } d : v \to w \text{ is a tuple of embeddings;}$$

$$\begin{array}{ll} e(d(x_s)) \stackrel{e}{=} (e \circ d)(x_s), & \text{if } d : s \to t \text{ and } e : t \to u \text{ are embeddings;} \\ id_s(x_s) = x_s, & \text{if } s \text{ is a sort;} \\ \mathrm{def}\, f(x_w), & \text{if } f : w \to s \text{ is a total function symbol;} \\ e(x_s) \stackrel{e}{=} e(y_s) \Rightarrow x_s \stackrel{e}{=} y_s, & \text{if } e : s \to t \text{ is an embedding.} \end{array}$$

Fig. 2. Axioms associated to an enriched CASL signature

associated partial morphism is total, i.e. iff the associated monomorphism is an isomorphism. Total monomorphisms double as embeddings, and partial function symbols with terminal codomain (taken modulo isomorphy of terminal objects) as predicate symbols.

Th_{enr} is essentially a left adjoint for U (where 'essentially' means that the factorizing lex functor in the universal property is unique only up to a natural isomorphism). In particular, Th_{enr} preserves colimits up to equivalence. Together with minor considerations concerning actual preservation of limits as opposed to preservation up to equivalence, this implies

Theorem 10. $\mathbf{Mod}_{enr}$ *has the amalgamation property.*

This is, of course, rather more easily proved directly. However, the representation Th_{enr} is not only of independent interest, but is also needed for the proof of definitional completeness. In fact,

Theorem 11. Th_{enr} *reflects equivalences.*

The proof relies on two facts about deduction in $\mathsf{Spec}(\Sigma) = \mathcal{S}$:

Definition 12. A term α in $\mathcal{S}$ is said to *reduce* to an embedding d in Σ if all symbols in α are embeddings, and their composite is d. Reduction to a symbol is defined analogously, using also the actions of the sort category.

Lemma 1. *Let α and β be terms, and let $h : w \to s$ be a symbol in $\mathcal{S}$ such that $\alpha \stackrel{e}{=} \alpha \vdash \beta \stackrel{e}{=} h(x_w)$. Then β reduces to h.*

Lemma 2. *If $\vdash \mathrm{def}\, f(x_w)$ in $\mathcal{S}$ for a function symbol $f : w \to s$ in Σ, then f is total.*

In a nutshell, this means that the only terms that are provably equal to symbols are the obvious candidates, and that only total function symbols are provably total (note that, even for standard CASL signatures, there are provably total *terms* that cannot be expressed using only total function symbols!).

By Theorems 7 and 11, $\mathbf{Mod}_{enr}$ reflects equivalences. Since it is trivial to show that a signature morphism σ is bijective on sorts if $\mathbf{Mod}_{enr}(\sigma)$ is an isomorphism, this implies

Corollary 13. $\mathbf{Mod}_{enr}$ *is definitionally complete.*

In particular, we now have a necessary and sufficient criterion for amalgamability in **CASLsign**:

Corollary 14. *A cocone in* **CASLsign** *is amalgamable iff its image under* Enr *is a colimit in* **enrCASLsign**.

5 Conclusions and Future Work

We have constructed a representation of the CASL institution (the extension to sentences, disregarded here, is straightforward) in enriched CASL, an institution with a category of enriched signatures that has the amalgamation property. This makes a number of results (e.g. concerning normal forms and proof systems) about institution independent specification languages applicable to CASL.

In more detail, we have modified Reynolds' approach [23] to subsorting via sort categories by using actions of the sort category on function and predicate symbols. In this way we elegantly deal with the problems of both overloading and amalgamation. Moreover, the associated logic admits a reduction to partial conditional equational logic. For the latter, we provide a sound and complete proof system, extending and simplifying the work of [8].

Typically, the use of enriched CASL will be as follows. Specifications are written in ordinary CASL. In situations where the user inputs a certain combination of signatures (e.g. when writing an instantiation of a parameterized specification), the natural requirement will be to check whether this combination becomes a colimit in enriched CASL, thus guaranteeing amalgamability of models. At this stage, enriched CASL remains completely hidden from the user. In contrast to this, there are also situations where a combination of signatures is automatically produced by a tool (e.g. during a proof in a development graph or during static analysis of architectural specifications). In these situations, it is advisable to use colimits that result in properly enriched signatures as intermediate results rather than discard them. Theorem proving in the enriched CASL logic is eased by the fact that this logic can be embedded into partial first-order logic with equality in much the same way as the CASL logic.

Moreover, we have shown that enriched CASL is definitionally complete, which means that we can expect amalgamation only for combinations of signatures that are actually colimits in the enriched signature category. Thanks to this characterization of a model-theoretic condition by a syntactic condition concerning signature combination, the model-theoretic and the static architectural semantics of [27] essentially coincide. Algorithms related to the actual computation of colimits of enriched signatures (a prerequisite for the development of tools for architectural specifications and proof support) are discussed in the forthcoming [17].

Future lines of research include the generalization of the techniques developed in this work to arbitrary institutions, possibly resulting in a generic procedure for 'making institutions amalgamable'. In particular, it is plausible to assume that the institutions with unnamed universes mentioned in the introduction can

— in analogy to the extension of unsorted to multisorted logic — be made amalgamable by introducing 'multiple universes'.

Moreover, we conjecture that not only the amalgamation property, but also the Craig interpolation property (in the weakened form valid in multisorted algebra), which fails in standard CASL, holds in enriched CASL. This property is required e.g. by the institution independent proof calculus of [6]. Finally, the representation of enriched signatures as left exact categories (which form a cartesian closed category) may provide the proper framework for a semantics of CASL architectural terms with bound variables [10] in the spirit of [27].

Acknowledgements. Partial support by the CoFI Working Group (ESPRIT WG 29432) is gratefully acknowledged. Moreover, we wish to thank Bartek Klin and Piotr Hoffman for collaboration and Maura Cerioli for pointing out the lack of amalgamation for CASL models.

References

[1] J. Adámek, H. Herrlich, and G. E. Strecker, *Abstract and concrete categories*, 2nd ed., Wiley Interscience, New York, 1990.
[2] D. Aspinall, *Types, subtypes, and ASL+*, Recent Trends in Data Type Specification, LNCS, vol. 906, Springer, 1994, pp. 116–131.
[3] H. Baumeister, *Relations between abstract datatypes modeled as abstract datatypes*, Ph.D. thesis, Universität des Saarlandes, 1998.
[4] M. Bidoit, M. V. Cengarle, and R. Hennicker, *Proof systems for structured specifications and their refinements*, Algebraic Foundations of Systems Specification (E. Astesiano et al., eds.), Springer, 1999, pp. 385–433.
[5] F. Borceux, *Handbook of categorical algebra 1*, Cambridge, 1994.
[6] T. Borzyszkowski, *Logical systems for structured specifications*, Theoret. Comput. Sci., to appear.
[7] M. Cerioli, A. Haxthausen, B. Krieg-Brückner, and T. Mossakowski, *Permissive subsorted partial logic in* CASL, Algebraic Methodology and Software Technology, LNCS, vol. 1349, Springer, 1997, pp. 91–107.
[8] I. Claßen, M. Große-Rhode, and U. Wolter, *Categorical concepts for parameterized partial specifications*, Math. Struct. Comput. Sci. **5** (1995), 153–188.
[9] CoFI, *The Common Framework Initiative for algebraic specification and development, electronic archives*, accessible by WWW[2] and FTP[3].
[10] CoFI Language Design Task Group, CASL *Summary, version 1.0*, Documents/CASL/Summary, in [9], July 1999.
[11] R. Diaconescu, J. Goguen, and P. Stefaneas, *Logical support for modularisation*, Logical Environments, Cambridge, 1993, pp. 83–130.
[12] H. Ehrig and M. Große-Rhode, *Functorial theory of parameterized specifications in a general specification framework*, Theoret. Comput. Sci. **135** (1994), 221–266.
[13] H. Ehrig and B. Mahr, *Fundamentals of algebraic specification 2*, Springer, 1990.
[14] J. Goguen and R. Burstall, *Institutions: Abstract model theory for specification and programming*, J. ACM **39** (1992), 95–146.

[2] http://www.brics.dk/Projects/CoFI
[3] ftp://ftp.brics.dk/Projects/CoFI

[15] R. Harper, F. Honsell, and G. D. Plotkin, *A framework for defining logics*, J. ACM **40** (1993), 143–184.

[16] B. Klin, An *implementation of static semantics for architectural specifications in* Casl *(in Polish)*, Master's thesis, Warsaw University, 2000.

[17] B. Klin, P. Hoffman, A. Tarlecki, T. Mossakowski, and L. Schröder, *Checking amalgamability conditions for* Casl *architectural specifications*, work in progress; see also [16].

[18] F. W. Lawvere, *Functorial semantics of algebraic theories*, Proc. Natl. Acad. Sci. USA **50** (1963), 869–872.

[19] S. Mac Lane, *Categories for the working mathematician*, Springer, 1997.

[20] M. Makkai and A. M. Pitts, *Some results on locally finitely presentable categories*, Trans. Amer. Math. Soc. **299** (1987), 473–496.

[21] T. Mossakowski, S. Autexier, and D. Hutter, *Extending development graphs with hiding*, Fundamental Approaches to Software Engineering, LNCS, vol. 2029, Springer, 2001, pp. 269–283.

[22] P. D. Mosses, Casl*: A guided tour of its design*, Workshop on Abstract Datatypes, LNCS, vol. 1589, Springer, 1999, pp. 216–240.

[23] J. C. Reynolds, *Using category theory to design implicit conversions and generic operators*, Semantics-Directed Compiler Generation, LNCS, vol. 94, Springer, 1980, pp. 211–258.

[24] D. Sannella, S. Sokołowski, and A. Tarlecki, *Towards formal development of programs from algebraic specifications: Parameterisation revisited*, Acta Inform. **29** (1992), 689–736.

[25] D. Sannella and A. Tarlecki, *Specifications in an arbitrary institution*, Inform. and Comput. **76** (1988), 165–210.

[26] L. Schröder, T. Mossakowski, A. Tarlecki, P. Hoffman, and B. Klin, *Amalgamation in the semantics of* Casl, work in progress; preprint to be made available[4].

[27] ———, *Semantics of architectural specifications in* Casl, Fundamental Approaches to Software Engineering, LNCS, vol. 2029, Springer, 2001, pp. 253–268.

[4] http://www.tzi.de/cofi/papers/amalg.ps

Lower Bounds for the Weak Pigeonhole Principle Beyond Resolution

Albert Atserias*, María Luisa Bonet**, and Juan Luis Esteban***

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya, Barcelona
{atserias,bonet,esteban}@lsi.upc.es†

Abstract. We work with an extension of Resolution, called Res(2), that allows clauses with conjunctions of two literals. In this system there are rules to introduce and eliminate such conjunctions. We prove that the weak pigeonhole principle PHP_n^{cn} and random unsatisfiable CNF formulas require exponential-size proofs in this system. This is the strongest system beyond Resolution for which such lower bounds are known. As a consequence to the result about the weak pigeonhole principle, Res(log) is exponentially more powerful than Res(2). Also we prove that Resolution cannot polynomially simulate Res(2), and that Res(2) does not have feasible monotone interpolation solving an open problem posed by Krajíček.

1 Introduction

The pigeonhole principle, PHP_n^{n+1}, expresses that it is not possible to have a one-to-one mapping from $n+1$ pigeons to n holes. This simple principle is central to many mathematical arguments. Since it can be formalized in propositional logic, it is natural to ask in which propositional proof systems such a principle can be proved in polynomial-size, with respect to the size of the encoding.

A fair amount of information is known about sizes of proofs of PHP_n^{n+1} in various proof systems. Haken [12] proved that this principle requires exponential-size proofs in Resolution. His proof techniques were later extended and simplified [4,5]. Also Beame et al. [2] proved that PHP_n^{n+1} requires exponential-size proofs in bounded-depth Frege systems. Regarding upper bounds, Buss [8] gave polynomial-size proofs of PHP_n^{n+1} in unrestricted Frege systems.

The pigeonhole principle can be formulated in more general terms, allowing the number of pigeons to be greater than $n+1$. We call this principle weak pigeonhole principle, or PHP_n^m, when the number of pigeons m is at least $2n$. The proof techniques of Haken where extended in [9] to prove that $\mathrm{PHP}_n^{n^{2-\epsilon}}$ requires exponential-size proofs in Resolution. A very intriguing and often studied

* Supported by the CUR, Generalitat de Catalunya, through grant 1999FI 00532.
** Partially supported by MEC through grant PB98-0937-C04 (FRESCO project) and CICYT TIC 98-0410-C02-01.
*** Partially supported by MEC through grant PB98-0937-C04 (FRESCO project)
† Partially supported by ALCOM-FT, IST-99-14186.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 1005–1016, 2001.

open problem is to prove exponential-size lower bounds for Resolution proofs of $\mathrm{PHP}_n^{n^c}$, if that can be done at all. As a contrast, the techniques of [2] for proving lower bounds for the pigeonhole principle in bounded-depth Frege systems can only prove lower bounds for PHP_n^{n+c}, and it is again open whether lower bounds can be proved when the number of pigeons in greater than $n+c$. Regarding upper bounds, it is known that PHP_n^{2n} has quasipolynomial-size proofs in bounded-depth Frege [15,14].

We work with the proof system Res(2), proposed by Krajíček [13], that can be viewed either as an extension of Resolution, or as a restriction of bounded-depth Frege. In this system the clauses do not only contain literals, but can also have conjunctions of two literals. The resolution rule gets modified to be able to eliminate a conjunction of two literals from a clause. We prove that PHP_n^{cn} requires exponential-size proofs in Res(2). This is, to our knowledge, the first lower bound proof for the weak pigeonhole principle in a subsystem of bounded-depth Frege that extends Resolution. We note that the quasipolynomial upper bound for bounded-depth Frege mentioned above can be carried over in depth-0.5 LK [14], which is equivalent to Res(log) (the analogue of Res(2) when we allow conjunctions of up to polylog literals). As a consequence of our lower bound, there is an exponential separation between Res(2) and Res(log).

Combining our techniques with those of [3], we also obtain an exponential-size lower bound for Res(2)-refutations of random unsatisfiable k-CNF formulas with clause density near the threshold. Again, this is the strongest system beyond Resolution for which such a lower bound is known. This result may be considered as a first step towards proving them hard for bounded-depth Frege.

Another important question to ask is whether Res(2) is more powerful than Resolution. Here we prove that Resolution cannot polynomially simulate Res(2), and therefore Res(2) is superpolynomially more efficient than Resolution. As a corollary, we see that Res(2) does not have the feasible monotone interpolation property, solving this way a conjecture of Krajíček [13].

Another motivation for working with the system Res(2) is to see how useful it can be in automated theorem proving. Given that it is more efficient than Resolution (at least there is a superpolynomial separation), it might be a good idea to try to find good heuristics to find proofs in Res(2) to be able to use it as a theorem prover.

2 Definitions and Overview of the Lower Bound Proof

A *k-term* is a conjunction of up to k literals. A *k-disjunction* is an (unbounded fan-in) disjunction of k-terms. If F is a k-disjunction, a 1-term of F is also called a *free-literal*. The refutation system Res(k), defined by Krajíček [13], works with k-disjunctions. There are three inference rules in Res(k): Weakening, $\wedge$-Introduction, and Cut

$$\frac{A}{A \vee \bigwedge_{i\in I} l_i} \qquad \frac{A \vee \bigwedge_{i\in I} l_i \qquad B \vee \bigwedge_{i\in J} l_i}{A \vee B \vee \bigwedge_{i\in I\cup J} l_i} \qquad \frac{A \vee \bigwedge_{i\in I} l_i \qquad B \vee \bigvee_{i\in I} \bar{l}_i}{A \vee B}$$

where A and B are k-disjunctions, I, J are sets of indices such that $|I \cup J| \leq k$, and the l_i's are literals. As usual, if l is a literal, $\bar{l}$ denotes its negation. Observe that Res(1) coincides with Resolution with the Weakening rule. The size of a Res(k)-refutation is the number of symbols in it. Mainly, we will work with Res(2).

Let $G = (U \cup V, E)$ be a bipartite graph on the sets U and V of cardinality m and n respectively, where $m > n$. The G-PHP$_n^m$, defined by Ben-Sasson and Wigderson [5], states that there is no matching of U into V. For every edge $(u, v) \in E$, let $x_{u,v}$ be a propositional variable meaning that u is mapped to v. The principle is then formalized as the conjunction of the following set of clauses:

$$x_{u,v_1} \vee \cdots \vee x_{u,v_r} \qquad u \in U,\ N_G(u) = \{v_1, \ldots, v_r\} \tag{1}$$

$$\bar{x}_{u,v} \vee \bar{x}_{u',v} \qquad v \in V,\ u, u' \in N_G(v),\ u \neq u'. \tag{2}$$

Here, $N_G(w)$ denotes the set of neighbors of w in G. Observe that if G is the complete bipartite graph K_n^m, then G-PHP$_n^m$ coincides with the usual pigeonhole principle PHP$_n^m$. It is easy to see that a lower bound for the size of Res(2)-refutations of G-PHP$_n^m$ implies the same lower bound for the size of Res(2)-refutations of PHP$_n^m$.

Ben-Sasson and Wigderson proved that whenever G is expanding in a sense defined next, every Resolution refutation of G-PHP$_n^m$ must contain a clause with many literals. We observe that this result is not unique to Resolution and holds in a more general setting. Before we state the precise result, let us recall the definition of expansion:

Definition 1. [5] *Let $G = (U \cup V, E)$ be a bipartite graph where $|U| = m$, and $|V| = n$. For $U' \subset U$, the* boundary *of U', denoted by $\partial U'$, is the set of vertices in V that have exactly one neighbor in U'; that is, $\partial U' = \{v \in V : |N(v) \cap U'| = 1\}$. We say that G is* (m, n, r, f)-expanding *if every subset $U' \subseteq U$ of size at most r is such that $|\partial U'| \geq f \cdot |U'|$.*

The proof of the following statement is the same as in [5] for Resolution.

Theorem 1. [5] *Let $\mathcal{S}$ be a sound refutation system with all rules having fan-in at most two. Then, if G is (m, n, r, f)-expanding, every $\mathcal{S}$-refutation of G-PHP$_n^m$ must contain a formula that involves at least $rf/2$ distinct literals.*

With these definitions, we are ready to outline the argument of the lower bound proof. In section 3, we will prove the existence of a bipartite graph $G = (U \cup V, E)$ with $|U| = cn'$ and $|V| = n'$ such that if we remove a small random subset of nodes from V, and the corresponding edges, the resulting graph is (m, n, r, f)-expanding for certain m, n, r and f. Then we will argue that G-PHP$_{n'}^{cn'}$ requires exponential-size Res(2)-refutations as follows. Assume, for contradiction, that Π is a small refutation of G-PHP$_{n'}^{cn'}$. We say that a 2-disjunction in Π is large if it contains at least $d = rf/2$ distinct literals. We apply a random restriction ρ_1 to the refutation such that for every large C, either $C|_{\rho_1}$ contains many free

literals, or the total number of literals in $C|_{\rho_1}$ is less than d. Then we extend ρ_1 to a new random restriction $\rho \supseteq \rho_1$ that knocks out all those large C such that $C|_{\rho_1}$ contains many free literals, ignoring those that are not free. After applying ρ, we obtain a refutation of $G(\rho)$-PHP$_n^m$ where all 2-disjunctions have less than $rf/2$ literals and $G(\rho)$ is (m, n, r, f)-expanding. This contradicts Theorem 1.

3 Random Graphs and Restrictions

In this section we will prove the existence of a bipartite graph G as claimed in Section 2. The principle G-PHP$_n^m$ will require exponential size Res(2)-proofs.

Let $\mathcal{G}(m, n, p)$ denote the distribution on bipartite graphs on sets U and V of sizes m and n respectively, with edge probability p independently for each edge.

Lemma 1. *If G is drawn from $\mathcal{G}(m, n, p)$, then* $\Pr[\forall v \in V : mp/2 < \deg_G(v) < 2mp] \geq 1 - 2ne^{-\frac{mp}{8}}$.

Proof: Fix a vertex $v \in V$. Then, $\deg_G(v) \sim \text{Bin}(m, p)$, so that $\text{E}[\deg_G(v)] = mp$. By Chernoff bounds, $\Pr[\deg_G(v) \geq 2mp] \leq e^{-mp/3}$ and $\Pr[\deg_G(v) \leq mp/2] \leq e^{-mp/8}$. By a union bound, $\Pr[\exists v \in V : \deg_G(v) \leq mp/2 \vee \deg_G(v) \geq 2mp] \leq ne^{-mp/3} + ne^{-mp/8} \leq 2ne^{-mp/8}$, and so $\Pr[\forall v \in V : mp/2 < \deg_G(v) < 2mp] \geq 1 - 2ne^{-mp/8}$. □ (lemma 1)

Lemma 2. *Let $m = kn$, $p = 48k\ln(m)/m$, $\alpha = 1/mp$ and $f = np/6$. Let G be drawn from $\mathcal{G}(m, n, p)$. Then,* $\Pr[G \text{ is } (m, n, \alpha m, f)\text{-expanding}] \geq 1/2$.

Proof: Fix $U' \subseteq U$ of size $s \leq \alpha m$, and $v \in V$. Then, $\Pr[v \in \partial U'] = sp(1-p)^{s-1}$. Let $q = \Pr[v \in \partial U']$. Let X_v be the indicator random variable for the event that $v \in \partial U'$. Then, $|\partial U'| = \sum_{v \in V} X_v$. Observe that X_v and $X_{v'}$ are independent whenever $v \neq v'$. Hence, $|\partial U'| \sim \text{Bin}(n, q)$, so that $E[|\partial U'|] = nq$. By Chernoff bound, $\Pr[|\partial U'| \leq nq/2] \leq e^{-nq/8}$. On the other hand, $nq = nsp(1-p)^{s-1} \geq snp(1-p)^{\alpha m}$. Moreover, $(1-p)^{\alpha m} = (1-p)^{1/p}$ approaches $1/e$ for sufficiently large m. Therefore, $nq \geq snp/3$. It follows that $nq/2 \geq sf$ and $e^{-nq/8} \leq e^{-snp/24}$. We conclude that $\Pr[|\partial U'| < f \cdot |U'|] \leq \Pr[|\partial U'| \leq nq/2] \leq e^{-nq/8} \leq e^{-snp/24}$. Finally, we bound the probability that G is not $(m, n, \alpha m, f)$-expanding by

$$\sum_{s=1}^{\alpha m} \binom{m}{s} e^{-snp/24} \leq \sum_{s=1}^{\alpha m} m^s e^{-snp/24} \leq \sum_{s=1}^{\alpha m} (me^{-np/24})^s. \tag{3}$$

Recall that $p = 48k\ln(m)/m$ and $m = kn$. So $me^{-np/24} \leq me^{-2\ln(m)} = m^{-1} < 1/4$. Hence the sum in (3) is bounded by $\sum_{s=1}^{\infty} \frac{1}{4^s} \leq \frac{1}{2}$. □ (lemma 2)

Let G be a fixed bipartite graph on $\{1, \ldots, m\}$ and $\{1, \ldots, n\}$. A *restriction* (for G) is a sequence of pairs $\rho = ((u_1, v_1), \ldots, (u_r, v_r))$ such that $(u_i, v_i) \in E(G)$, and all v_i's are distinct. We let $R_r(G)$ be the set of restrictions of length r. We define a distribution $\mathcal{R}_r(G)$ on $R_r(G)$ as follows: Let $V_0 = \{1, \ldots, n\}$; for every $i \in \{1, \ldots, r\}$ in increasing order, choose a hole v_i uniformly at random in

V_{i-1}, choose a pigeon u_i uniformly at random in $N_G(v_i)$, and let $V_i = V_{i-1} - \{v_i\}$. The final restriction is $((u_1, v_1), \ldots, (u_r, v_r))$.

We define a distribution $\mathcal{D}(m, n, p, r)$ on the set of pairs (G, ρ) with $\rho \in R_r(G)$: the graph G is drawn from $\mathcal{G}(m, n + r, p)$ first, and then ρ is drawn from $\mathcal{R}_r(G)$. In other words, if (H, π) is a fixed pair with $\pi \in R_r(H)$, then $\Pr[G = H \wedge \rho = \pi] = p^{e(H)}(1-p)^{m(n+r)-e(H)}|R_r(H)|^{-1}$.

If G is a bipartite graph on $\{1, \ldots, m\}$ and $\{1, \ldots, n+r\}$, and ρ a restriction $((u_1, v_1), \ldots, (u_r, v_r)) \in R_r(G)$, then $G(\rho)$ denotes the graph that results from deleting $v_1, \ldots, v_r$ from G, and renaming nodes in an order-preserving way. With this definitions we are ready to prove:

Lemma 3. *Let $m = kn$, $p = 48k\ln(m)/m$, $\alpha = 1/mp$ and $f = np/6$. Let (G, ρ) be drawn from $\mathcal{D}(m, n, p, r)$. Then, $\Pr[G(\rho)$ is $(m, n, \alpha m, f)$-expanding$] \geq 1/2$.*

Proof: Let A be the event that $G(\rho)$ is $(m, n, \alpha m, f)$-expanding. Let $S = \{R \subseteq \{1, \ldots, n + r\} : |R| = r\}$. Then, $\Pr[A] = \sum_{R \in S} \Pr[A \mid \mathrm{ran}(\rho) = R] \Pr[\mathrm{ran}(\rho) = R]$. The proof that $\Pr[A \mid \mathrm{ran}(\rho) = R] \geq 1/2$ is the same as the proof of Lemma 2 replacing V by $V - R$. The result follows. □ (lemma 3)

Lemma 4. *Let $m = kn$, $p = 48k\ln(m)/m$, $\alpha = 1/mp$ and $f = np/6$. For every $r \leq n$, there exists a bipartite graph H on $\{1, \ldots, m\}$ and $\{1, \ldots, n + r\}$ such that the following two properties hold:*

(i) $mp/2 \leq \deg_H(v) \leq 2mp$ *for every* $v \in \{1, \ldots, n + r\}$,
(ii) $\Pr[H(\rho)$ *is* $(m, n, \alpha m, f)$*-expanding*$] \geq 1/3$,

when ρ is drawn from $\mathcal{R}_r(H)$.

Proof: Let (G, ρ) be drawn from $\mathcal{D}(m, n, p, r)$. According to Lemmas 3, we have We have that $\Pr[G(\rho)$ is $(m, n, \alpha m, f)$-expanding$] \geq 1/2$. Moreover, $\Pr[\forall v \in V : mp/2 < \deg_G(v) < 2mp] \geq 1 - (n+r)e^{-mp/9} \geq 5/6$ by Lemma 1. Let $E(G, \rho)$ be the event that $G(\rho)$ is expanding *and* every right-node in G has degree between $mp/2$ and $2mp$. Combining both equations above we have that $\Pr[E(G, \rho)] \geq 1/3$. On the other hand, $\Pr[E(G, \rho)] = \sum_H \Pr[E(G, \rho) \mid G = H] \Pr[G = H]$ where H ranges over all bipartite graphs on m and $n + r$ nodes. Therefore, there exists some fixed H such that $\Pr[E(G, \rho) \mid G = H] \geq 1/3$. Moreover, $\Pr[E(G, \rho) \mid G = H]$ equals $\Pr[E(H, \pi)]$ when π is drawn from $\mathcal{R}_r(H)$. Finally, since this probability is strictly positive, it must be the case that H satisfies property (i) in the lemma since it is independent of π. □ (lemma 4)

4 The Lower Bound Argument

Before we start the lower bound proof, we need a normal form for Res(2)-refutations of G-PHP$_n^m$. We claim that every Res(2)-refutation of G-PHP$_n^m$ can be turned into a Res(2)-refutation of similar size in which no 2-term is of the form $x_{u,v} \wedge x_{u',v}$ with $u \neq u'$. To check this, observe that such a 2-term must have been introduced at some point by the rule of $\wedge$-introduction with clauses, say, $A \vee x_{u,v}$ and $B \vee x_{u',v}$. Cutting them with the axiom $\bar{x}_{u,v} \vee \bar{x}_{u',v}$ we get $A \vee B$ that can be used to continue the proof because it subsumes $A \vee B \vee (x_{u,v} \wedge x_{u',v})$.

Theorem 2. *Let $c > 1$ be a constant. For all sufficiently large n, every* Res(2)*-refutation of* PHP_n^{cn} *has size at least* $e^{n/(\log n)^{14}}$.

Proof: Let $k = c+1$, $r = n/c$, $n' = n+r$, and $m = kn = cn'$. Let $G = (U \cup V, E)$ with $|U| = m$ and $|V| = n+r$ be the bipartite graph of Lemma 4. We show that every Res(2)-refutation of G-PHP has size at least $e^{n/(\log n)^{14}}$. This will imply the Theorem since a Res(2)-refutation of $\mathrm{PHP}_{n'}^{cn'}$ gives a Res(2)-refutation of G-PHP of no bigger size.

Let us assume, for contradiction, that G-PHP has a Res(2)-refutation Π of size $S < e^{n/(\log n)^{14}}$. Let C be a 2-disjunction, and let $(u, v) \in E(G)$. We let $C|_{(u,v)}$ be the result of assigning $x_{u,v} = 1$ and $x_{u',v} = 0$ for every $u' \in N_G(v) - \{u\}$ to C, and simplifying as much as possible. This includes replacing subformulas of the form $l \vee (l \wedge l')$ by l, and subformulas of the form $\bar{l} \vee (l \wedge l')$ by $\bar{l} \vee l'$ in some specified order; here l and l' are literals. Given a restriction $\rho = ((u_1, v_1), \ldots, (u_r, v_r))$, we let $C|_\rho$ be the result of applying $(u_1, v_1), \ldots, (u_r, v_r)$ successively in this order. For every $i \in \{1, \ldots, r\}$, we let $\rho_i = ((u_1, v_1), \ldots, (u_i, v_i))$. We say that C is *large* if it contains at least $d = n/12$ distinct literals; otherwise, C is *small*. We say that C is *wide* if it contains at least $s = n/(\log n)^5$ free literals; otherwise, C is *narrow*. We say that a pair $(u, v) \in E(G)$ *hits* C if either $x_{u,v}$ occurs positively in C, or $x_{u',v}$ occurs negatively in C for some $u' \in N_G(v) - \{u\}$. Equivalently, (u, v) hits C if it sets some literal of C to 1. If the literal is free, it knocks out the 2-disjunction. If the literal is part of a conjunction, it will locally create a free literal. In general, we say that $(u, v) \in E(G)$ *knocks* C if $C|_{(u,v)} \equiv 1$. We say that $(u, v) \in E(G)$ is a *bad pair* for C if it does not knock it and there exists $u' \in N_G(v) - \{u\}$ such that (u', v) knocks C. A bad pair may or may not be a hit.

In all probabilities that follow, ρ is drawn from the distribution $\mathcal{R}_r(G)$. Our main goal is to prove that the probability that a fixed 2-disjunction C of Π remains large is exponentially small; that is, we aim for a proof that

$$\Pr[C|_\rho \text{ is large}] \leq e^{-n/(\log n)^{13}}. \tag{4}$$

This will suffice because then $\Pr[\exists C \in \Pi : C|_\rho \text{ is large}] \leq Se^{-n/(\log n)^{13}} < e^{n/(\log n)^{14}} e^{-n/(\log n)^{13}} < 1/3$, and also $\Pr[G(\rho) \text{ not } (m, n, \alpha m, f)\text{-expanding}] \leq 2/3$ by Lemma 4. This means that there exists a restriction $\rho \in R_r(G)$ such that $G(\rho)$ is $(m, n, \alpha m, f)$-expanding and every 2-disjunction in $\Pi|_\rho$ has less than $d = \alpha m f/2$ literals. This is a contradiction with Theorem 1.

For $i \in \{1, \ldots, r\}$, let A_i be the event that $C|_{\rho_i}$ is large, and let B_i be the event that $C|_{\rho_i}$ is narrow. Recall that $\rho_i = ((u_1, v_1), \ldots, (u_i, v_i))$. Then,

$$\Pr[C|_\rho \text{ is large}] \leq \Pr\left[A_r \wedge \bigvee_{j \geq r/2} B_j\right] + \Pr\left[A_r \wedge \bigwedge_{j \geq r/2} \overline{B_j}\right] \leq$$

$$\leq \sum_{j=r/2}^{r} \Pr[A_j \wedge B_j] + \Pr\left[A_r \wedge \bigwedge_{j \geq r/2} \overline{B_j}\right].$$

We will show that every term in this expression is exponentially small. The bound on terms of the form $\Pr[A_j \wedge B_j]$ will be proven in Lemma 6. For the last term, we use an argument similar in spirit to the one by Beame and Pitassi [4]:

Lemma 5. $\Pr\left[A_r \wedge \bigwedge_{j \geq r/2} \overline{B_j}\right] \leq e^{-n/(\log n)^8}$.

Proof: Let S_i be the indicator random variable for the event that (u_i, v_i) knocks $C|_{\rho_{i-1}}$. Then,

$$\begin{aligned}
\Pr\left[A_r \wedge \bigwedge_{j \geq r/2} \overline{B_j}\right] &\leq \Pr\left[\bigwedge_{i > r/2} S_i = 0 \wedge \bigwedge_{j \geq r/2} \overline{B_j}\right] = \\
&= \prod_{i > r/2} \Pr\left[S_i = 0 \wedge \bigwedge_{j \geq r/2} \overline{B_j} \;\middle|\; \bigwedge_{r/2 < j < i} S_j = 0\right] \leq \\
&\leq \prod_{i > r/2} \Pr\left[S_i = 0 \wedge \overline{B_{i-1}} \;\middle|\; \bigwedge_{r/2 < j < i} S_j = 0\right] \leq \\
&\leq \prod_{i > r/2} \Pr\left[S_i = 0 \;\middle|\; \overline{B_{i-1}} \wedge \bigwedge_{r/2 < j < i} S_j = 0\right].
\end{aligned}$$

Fix $i \in \{r/2+1, \ldots, r\}$ and let H be the set of holes that occur in a free literal of $C|_{\rho_{i-1}}$. Given that $\overline{B_{i-1}}$ holds, $C|_{\rho_{i-1}}$ is wide which means that there are at least s free literals. Therefore $|H| \geq s/2\Delta$, where $\Delta = 2mp$ is an upper bound on the right-degree of G. Moreover, every $v \in H$ gives a possible knock, and different holes give different knocks. The reason is the following: if $x_{u,v}$ is a free literal, then (u, v) is a knock; and if $\bar{x}_{u,v}$ is a free literal, then (u', v) is a knock for every $u' \in N_G(v) - \{u\}$, which is non-empty since the right-degree of G is at least two. Therefore,

$$\Pr\left[S_i = 1 \;\middle|\; \overline{B_{i-1}} \wedge \bigwedge_{r/2 < j < i} S_j = 0\right] \geq \frac{|H|}{\Delta(n + r - i + 1)} \geq \frac{s}{3\Delta^2 n}.$$

Therefore,

$$\Pr\left[A_r \wedge \bigwedge_{j \geq r/2} \overline{B_j}\right] \leq \left(1 - \frac{s}{3\Delta^2 n}\right)^{r/2} \leq e^{-\frac{sr}{6\Delta^2 n}} \leq e^{-n/(\log n)^8}. \quad \square \text{ (lemma 5)}$$

Lemma 6. *Let j be such that $r/2 \leq j \leq r$. Then,* $\Pr[A_j \wedge B_j] \leq e^{-n/(\log n)^{11}}$.

Proof: Recall that A_j is the event that $C|_{\rho_j}$ is large, and B_j is the event that $C|_{\rho_j}$ is narrow. We let S_i be the indicator random variable for the event that (u_i, v_i) hits $C|_{\rho_{i-1}}$, where $\rho_{i-1} = ((u_1, v_1), \ldots, (u_{i-1}, v_{i-1}))$. Let $S = \sum_{i=1}^{j} S_i$. Then, for every h,

$$\begin{aligned}
\Pr[A_j \wedge B_j] &= \Pr[A_j \wedge B_j \wedge S < h] + \Pr[A_j \wedge B_j \wedge S \geq h] \leq \\
&\leq \Pr[A_j \wedge S < h] + \Pr[A_j \wedge B_j \wedge S \geq h].
\end{aligned}$$

We show that each term in this expression is exponentially small. More precisely, we show that $\Pr[A_j \wedge S < h] \leq e^{-n/(\log n)^3}$ and $\Pr[A_j \wedge B_j \wedge S \geq h] \leq e^{-n/(\log n)^{10}}$ which is clearly enough to prove Lemma 6.

Claim 1 *Let $h = n/(\log n)^4$. Then, $\Pr[A_j \wedge S < h] \leq e^{-n/(\log n)^3}$.*

Proof: Let $Y = \{(a_1, \ldots, a_j) \in \{0,1\}^j : \sum_{i=1}^{j} a_i < h\}$. Observe that A_j implies A_i for every $i \leq j$ because if $C|_{\rho_j}$ is large, so is $C|_{\rho_i}$ for every $i \leq j$. Then,

$$\begin{aligned}
\Pr[A_j \wedge S < h] &= \Pr\left[\sum_{i=1}^{j} S_i < h \wedge A_j\right] = \sum_{\overline{a} \in Y} \Pr\left[\bigwedge_{i=1}^{j} S_i = a_i \wedge A_j\right] = \\
&= \sum_{\overline{a} \in Y} \prod_{i=1}^{j} \Pr\left[S_i = a_i \wedge A_j \;\middle|\; \bigwedge_{k=1}^{i-1} S_k = a_k\right] \leq \\
&\leq \sum_{\overline{a} \in Y} \prod_{i=1}^{j} \Pr\left[S_i = a_i \wedge A_{i-1} \;\middle|\; \bigwedge_{k=1}^{i-1} S_k = a_k\right] \leq \\
&\leq \sum_{\overline{a} \in Y} \prod_{i=1}^{j} \Pr\left[S_i = a_i \;\middle|\; A_{i-1} \wedge \bigwedge_{k=1}^{i-1} S_k = a_k\right].
\end{aligned}$$

Fix $i \in \{1, \ldots, j\}$. Let H be the set of holes that occur in $C|_{\rho_{i-1}}$. We have $|H| \geq d/2\Delta$ given that A_{i-1} holds. Again, $\Delta = 2mp$ is an upper bound to the right-degree of G. Moreover, every $v \in H$ gives a possible hit, and different holes give different hits (the reason is the same as in Lemma 5 for knocks). Therefore,

$$\Pr\left[S_i = 1 \;\middle|\; A_{i-1} \wedge \bigwedge_{k=1}^{i-1} S_k = a_k\right] \geq \frac{|H|}{\Delta(n + r - i + 1)} \geq \frac{d}{3\Delta^2 n}.$$

Since there are at least $j - h$ zeros in $(a_1, \ldots, a_j)$, we obtain

$$\begin{aligned}
\Pr\left[A_j \wedge S < h\right] &\leq \sum_{\overline{a} \in Y} \left(1 - \frac{d}{3\Delta^2 n}\right)^{j-h} \leq \sum_{i<h} \binom{j}{i} e^{-\frac{d(j-h)}{3\Delta^2 n}} \leq h j^h e^{-\frac{d(j-h)}{3\Delta^2 n}} \leq \\
&\leq \exp\left(-\frac{j-h}{36\Delta^2} + h\log(j) + \log(h)\right) \leq e^{-n/(\log n)^3}. \square(\text{claim 1})
\end{aligned}$$

Claim 2 $\Pr[A_j \wedge B_j \wedge S \geq h] \leq e^{-n/(\log n)^{10}}$.

Proof: During this proof we will drop the subindex j in A_j and B_j since it will always be the same. For every $i \in \{1, \ldots, r\}$, let $T_i \in \{\mathrm{k}, \mathrm{b}, \mathrm{n}\}$ be a random variable indicating whether (u_i, v_i) is a knock, a bad pair, or none of the previous respectively for $C|_{\rho_{i-1}}$. For $t \in \{\mathrm{k}, \mathrm{b}, \mathrm{n}\}$, let S_i^t be the indicator random variable for the event that $T_i = t$, and let $S^t = \sum_{i=1}^{j} S_i^t$. Thus, S^{k} is the number of knocks and S^{b} is the number of bad pairs of ρ_j.

Fix $\rho = ((u_1, v_1), \ldots, (u_r, v_r))$ such that $A \wedge B \wedge S \geq h$ holds under ρ. Observe that (u_i, v_i) does not knock $C|_{\rho_{i-1}}$ for any $i \in \{1, \ldots, j\}$ since $C|_{\rho_j}$ must be large. Hence, $S^{\mathrm{k}} = 0$ under ρ. Let $b = (h - s)/(2\Delta + 1)$. We now claim that $S^{\mathrm{b}} \geq b$. Suppose for contradiction that the number of bad pairs is less than b. Every bad pair (u_i, v_i) removes at most 2Δ free literals since at most those many literals about hole v_i may appear. Moreover, since there are no knocks, every hit (u_i, v_i) that is not a bad pair increases the number of free literals by at least one. The reason is that such a hit turns a conjunction into a free literal whose hole component must be different from v_i for otherwise a knock would be available; recall that the right-degree of G is at least three, and that conjunctions of the form $x_{u,v} \wedge x_{u',v}$ with $u \neq u'$ do not occur in the refutation. It follows that the number of free literals in $C|_{\rho_j}$ is at least $(S - S^{\mathrm{b}}) - 2\Delta S^{\mathrm{b}} > h - (2\Delta + 1)b = s$, a contradiction with the fact that B holds under ρ. We have proved that $\Pr[A \wedge B \wedge S \geq h] \leq \Pr[S^{\mathrm{k}} = 0 \wedge S^{\mathrm{b}} \geq b]$. The intuition behind why this probability is small is that every bad pair could have been a knock. This makes unlikely that ρ produces many bad pairs and no knocks. In what follows, we will prove this intuition using martingales.

For $t \in \{\mathrm{k}, \mathrm{b}, \mathrm{n}\}$ and $i \in \{1, \ldots, j\}$, let P_i^t denote the random variable $\Pr[T_i = t \mid \rho_0, \ldots, \rho_{i-1}]$. We define a martingale $X_0, \ldots, X_j$ with respect to $\rho_0, \ldots, \rho_j$ as follows: Let $X_0 = 0$, and $X_{i+1} = X_i + S_{i+1}^{\mathrm{b}} - P_{i+1}^{\mathrm{b}}$. Recall that S_{i+1}^{b} is the indicator random variable for the event that $T_{i+1} = \mathrm{b}$. So

$$\begin{aligned}\mathrm{E}[X_{i+1} \mid \rho_0, \ldots, \rho_i] &= (X_i + 1 - P_{i+1}^{\mathrm{b}}) \cdot P_{i+1}^{\mathrm{b}} + (X_i - P_{i+1}^{\mathrm{b}}) \cdot (1 - P_{i+1}^{\mathrm{b}}) = \\ &= (X_i - P_{i+1}^{\mathrm{b}})(P_{i+1}^{\mathrm{b}} + 1 - P_{i+1}^{\mathrm{b}}) + P_{i+1}^{\mathrm{b}} = X_i.\end{aligned}$$

Hence, $\{X_i\}_i$ is a martingale with respect to $\{\rho_i\}_i$. Observe also that $X_j = S^{\mathrm{b}} - \sum_{i=1}^{j} P_i^{\mathrm{b}}$. Similarly, we define $Y_0, \ldots, Y_j$ as follows: Let $Y_0 = 0$, and $Y_{i+1} = Y_i + S_{i+1}^{\mathrm{k}} - P_{i+1}^{\mathrm{k}}$. It is also easy to see that $\{Y_i\}_i$ is a martingale with respect to $\{\rho_i\}_i$. Again, $Y_j = S^{\mathrm{k}} - \sum_{i=1}^{j} P_i^{\mathrm{k}}$.

Subclaim 1 *$P_i^{\mathrm{k}}(\rho) \geq P_i^{\mathrm{b}}(\rho)/\Delta$ for every $\rho \in R_r(G)$ and $i \in \{1, \ldots, j\}$.*

Proof: Fix $i \in \{1, \ldots, j\}$ and $\rho = ((u_1, v_1), \ldots, (u_r, v_r))$. We want to show that $P_i^{\mathrm{k}}(\rho) \geq P_i^{\mathrm{b}}(\rho)/\Delta$. Define three sets as follows: $Q = \{(u, v) \in E(G) : v \notin \{v_1, \ldots, v_{i-1}\}\}$, $Q^{\mathrm{k}} = \{(u, v) \in Q : (u, v) \text{ is a knock for } C|_{\rho_{i-1}}\}$, and $Q^{\mathrm{b}} = \{(u, v) \in Q : (u, v) \text{ is a bad pair for } C|_{\rho_{i-1}}\}$. Observe that $P_i^{\mathrm{b}}(\rho) = |Q^{\mathrm{b}}| \cdot |Q|^{-1}$ and $P_i^{\mathrm{k}}(\rho) = |Q^{\mathrm{k}}| \cdot |Q|^{-1}$. On the other hand, every bad pair $(u, v) \in Q^{\mathrm{b}}$ gives a possible knock $(u', v) \in Q^{\mathrm{k}}$ by definition. Moreover, bad pairs with different hole components give different possible knocks. Grouping Q^{b} by holes, we have that $|Q^{\mathrm{k}}| \geq |Q^{\mathrm{b}}|/\Delta$. Consequently, $P_i^{\mathrm{k}}(\rho) \geq P_i^{\mathrm{b}}(\rho)/\Delta$ as required. □ (subclaim 1)

To complete the proof of claim 2 we will need the following form of Azuma's Inequality: Let $X_0, \ldots, X_n$ be a martingale such that $|X_i - X_{i-1}| \leq 1$; then, $\Pr[|X_n - X_0| \geq \lambda] \leq 2e^{-\lambda^2/n}$ for every $\lambda > 0$ [11]. Now,

$$\begin{aligned}\Pr[S^{\mathrm{k}} = 0 \wedge S^{\mathrm{b}} \geq b] = \Pr[S^{\mathrm{k}} = 0 \wedge S^{\mathrm{b}} \geq b \wedge X_j \geq b/2] + \\ + \Pr[S^{\mathrm{k}} = 0 \wedge S^{\mathrm{b}} \geq b \wedge X_j < b/2].\end{aligned}$$

The first summand is bounded by $\Pr[X_j \geq b/2] \leq 2e^{-b^2/4j}$ by Azuma's Inequality. The second summand is bounded by

$$\Pr\left[S^{\mathrm{k}} = 0 \wedge \textstyle\sum_{i=1}^{j} P_i^{\mathrm{b}} \geq b/2\right] \leq \Pr\left[S^{\mathrm{k}} = 0 \wedge \textstyle\sum_{i=1}^{j} P_i^{\mathrm{k}} \geq b/2\Delta\right] \leq$$
$$\leq \Pr[Y_j \leq -b/2\Delta] \leq 2e^{-b^2/4\Delta^2 j}.$$

The first inequality follows from Subclaim 1, and the third follows from Azuma's Inequality again. The addition of the two summands is then bounded by $e^{-n/(\log n)^{10}}$ as required. □ (claim 2 and lemma 6)

We are ready to complete the proof of our goal: equation (4). We have shown that

$$\Pr[C|_\rho \text{ large}] \leq \sum_{j=r/2}^{r} e^{-n/(\log n)^{11}} + e^{-n/(\log n)^{8}} \leq e^{-n/(\log n)^{13}}.$$ □ (theorem 2)

The most general form of the previous theorem gives a lower bound for PHP_n^m of $e^{\frac{n^9}{(m \log m)^8 \log^3 n}}$, using $r = n/8$, $h = n^3/((m \log m)^2 \log n)$ and $s = h/2$, therefore the best result is an exponential lower bound for $\mathrm{PHP}_n^{n^{9/8-\epsilon}}$.

Given that Res(log) and depth-0.5 LK are polynomially equivalent, and given that PHP_n^{2n} has quasipolynomial-size proofs in depth-0.5 LK [14], we obtain:

Corollary 1. *There is an exponential separation between* Res(2) *and* Res(log).

The model of random k-CNF formulas that we use is the one considered in [10,3]. The distribution is denoted $\mathcal{F}_m^{k,n}$ and consists in choosing m clauses of exactly k literals independently with replacement. The following result can be obtained by an argument similar to that of Theorem 2. The proof can be found in the complete version of the paper.

Theorem 3. *Let* $F \sim \mathcal{F}_{5n}^{3,n}$. *Almost surely,* Res(2)*-refutations of* F *require size* $2^{\Omega(n^{1/3}/(\log(n))^2)}$.

5 Separation between Res(2) and Resolution

In this section we prove that Resolution cannot polynomially simulate Res(2). More precisely, we prove that a certain Clique-Coclique principle, as defined by Bonet, Pitassi and Raz in [6], has polynomial-size Res(2)-refutations, but every Resolution refutation requires quasipolynomial size.

The Clique-Coclique principle that we use, $\mathrm{CLIQUE}_{k,k'}^n$, is:

$x_{i,1} \vee \cdots \vee x_{i,n}$	$1 \leq l \leq k$
$\bar{x}_{l,i} \vee \bar{x}_{l,j}$	$1 \leq l \leq k,\ 1 \leq i,j \leq n,\ i \neq j$
$\bar{x}_{l,i} \vee \bar{x}_{l',i}$	$1 \leq l,l' \leq k,\ 1 \leq i \leq n,\ l \neq l'$
$y_{1,i} \vee \cdots \vee y_{k',i}$	$1 \leq i \leq n$
$\bar{y}_{l,i} \vee \bar{y}_{l',i}$	$1 \leq l,l' \leq k',\ 1 \leq i \leq n,\ l \neq l'$
$\bar{x}_{l,i} \vee \bar{x}_{l',j} \vee \bar{y}_{t,i} \vee \bar{y}_{t,j}$	$1 \leq l,l' \leq k,\ 1 \leq t \leq k',\ 1 \leq i,j \leq n,\ l \neq l',\ i \neq j$

Theorem 4. *Let $k' < k \leq n$. If $\mathrm{PHP}^k_{k'}$ has Resolution refutations of size S, then $\mathrm{CLIQUE}^n_{k,k'}$ has $\mathrm{Res}(2)$-refutations of size Sn^c for some constant $c > 0$.*

The proof can be found in the complete version of the paper. We reduce the formula $\mathrm{CLIQUE}^n_{k,k'}$ to $\mathrm{PHP}^k_{k'}$ in Res(2). The meaning of variable $p_{i,j}$ is that pigeon i sits in hole j. We perform the following substitutions:

$$p_{i,j} \equiv \bigvee_{l=1}^{n} (x_{i,l} \wedge y_{i,l}) \qquad \bar{p}_{i,j} \equiv \bigvee_{l=1, j' \neq j}^{n} (x_{i,l} \wedge y_{j',l})$$

We will use the Monotone Interpolation Theorem for Resolution together with the following result of Alon and Boppana [1] establishing a lower bound to the size of monotone circuits that separate large cliques from small cocliques. In the following, $F(m,k,k')$ is the set of monotone functions that separate k-cliques from k'-cocliques on m nodes. Let $S^+(f)$ be the monotone circuit size of f.

Theorem 5. *If $f \in F(m,k,k')$, $3 \leq k' \leq k$ and $k\sqrt{k'} \leq m/(8 \log m)$, then*

$$S^+(f) \geq \frac{1}{8} \left(\frac{m}{4k\sqrt{k'} \log m} \right)^{(\sqrt{k'}+1)/2}.$$

Theorem 6. *Let $k = \sqrt{m}$ and $k' = (\log m)^2/8 \log\log m$. Then, (i) $\mathrm{CLIQUE}^m_{k,k'}$ has $\mathrm{Res}(2)$-refutations of size polynomial in m, and (ii) every Resolution refutation of $\mathrm{CLIQUE}^m_{k,k'}$ has size at least $\exp(\Omega((\log m)^2/\sqrt{\log\log m}))$.*

Proof: Regarding (i), we have that $k' \log k' \leq \frac{1}{4}(\log m)^2$, and so $2^{\sqrt{k' \log k'}} \leq m^{1/2} = k$. On the other hand, Buss and Pitassi [7] proved that $\mathrm{PHP}^k_{k'}$ has Resolution refutations of size polynomial in k whenever $k \geq 2^{\sqrt{k' \log k'}}$. Therefore, by Theorem 4, $\mathrm{CLIQUE}^m_{k,k'}$ has Res(2)-refutations of size polynomial in m. Regarding (ii), we apply the feasible monotone interpolation theorem for Resolution. We have $\log m/(3\sqrt{\log\log m}) \leq \sqrt{k'} \leq \log m$. Therefore, by Theorem 5, if $f \in F(m,k,k')$ is a monotone interpolant, then

$$S^+(f) \geq \frac{1}{8} \left(\frac{m}{4\sqrt{m}(\log m)^2} \right)^{\frac{\log m}{6\sqrt{\log\log m}}} \geq \frac{1}{8} \left(\frac{m}{m^{3/4}} \right)^{\frac{\log m}{6\sqrt{\log\log m}}},$$

which is $\exp(\Omega((\log m)^2/\sqrt{\log\log m}))$. □ (theorem 6)

As a corollary, we solve an open problem posed by Krajíček [13].

Corollary 2. Res(2) *does not have the feasible monotone interpolation property.*

6 Discussion and Open Problems

We proved that there is a quasipolynomial separation between Resolution and Res(2). It is an open question whether the separation could be exponential, or a quasipolynomial simulation of Res(2) by Resolution exists. It is important to

notice, that our lower bound for PHP would not follow from such a simulation. Indeed, the lower bound that would follow from that would be of the form 2^{n^ϵ}.

The previous separation was obtained using a lower bound for Resolution via the monotone interpolation theorem. It is open whether this separation (or a stronger one) could be obtained via the size-width trade-off [5]. It would also be interesting to see what would that mean in terms of possible size-width trade-offs for Res(2). We conjecture that Res(2) does not have a strong size-width trade-off. Notice that Res(log) does not have it. This is because (a) Res(log) is equivalent to depth-0.5 LK, (b) PHP_n^{2n} has quasipolynomial-size proofs in depth-0.5 LK [14], and (c) PHP_n^{2n} has $\Omega(n)$ width lower bounds for Res(log).

We extended the width lower bound technique beyond Resolution. A very interesting open question is whether the technique can be extended for Res(3), etc. This is related to the optimality of the Res(log) upper bound for PHP_n^{2n}.

References

1. N. Alon and R. Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7(1):1–22, 1987.
2. P. Beame, R. Impagliazzo, J. Krajíček, T. Pitassi, P. Pudlák, and A. Woods. Exponential lower bounds for PHP. In *STOC92*, pages 200–220, 1992.
3. P. Beame, R. Karp, T. Pitassi, and M. Saks. The efficiency of resolution and Davis-Putnam procedures. Submitted. Previous version in STOC'98, 1999.
4. P. Beame and T. Pitassi. Simplified and improved resolution lower bounds. In *FOCS96*, pages 274–282, 1996.
5. E. Ben-Sasson and A. Wigderson. Short proofs are narrow: Resolution made simple. In *STOC99*, pages 517–527, 1999. Revised version (2000).
6. M. Bonet, T. Pitassi, and R. Raz. Lower bounds for cutting planes proofs with small coefficients. *The Journal of Symbolic Logic*, 62(3):708–728, Sept. 1997.
7. S. Buss and T. Pitassi. Resolution and the weak pigeonhole principle. In *CSL: 11th Workshop on Computer Science Logic*. LNCS, Springer-Verlag, 1997.
8. S. R. Buss. Polynomial size proofs of the propositional pigeonhole principle. *The Journal of Symbolic Logic*, 52(4):916–927, Dec. 1987.
9. S. R. Buss and G. Turán. Resolution proofs on generalized pigeonhole principles. *Theoretical Computer Science*, 62(3):311–317, Dec. 1988.
10. V. Chvátal and E. Szemerédi. Many hard examples for resolution. *J. ACM*, 35(4):759–768, 1988.
11. G. Grimmet and D. Stirzaker. *Probability and Random Processes*. Oxford, 1982.
12. A. Haken. The intractability of resolution. *TCS*, 39(2-3):297–308, Aug. 1985.
13. J. Krajíček. On the weak PHP. To appear in Fundamenta Mathematicæ, 2000.
14. A. Maciel, T. Pitassi, and A. Woods. A new proof of the weak pigeonhole principle. In *STOC00*, pages 368–377, 2000.
15. J. B. Paris, A. J. Wilkie, and A. R. Woods. Provability of the pigeonhole principle and the existence of infinitely many primes. *The Journal of Symbolic Logic*, 53(4):1235–1244, 1988.

Time and Space Bounds for Reversible Simulation*

(Extended Abstract)

Harry Buhrman, John Tromp, and Paul Vitányi

CWI, Kruislaan 413, 1098 SJ Amsterdam,
The Netherlands, email{buhrman,tromp,paulv}@cwi.nl

Abstract. We prove a general upper bound on the tradeoff between time and space that suffices for the reversible simulation of irreversible computation. Previously, only simulations using exponential time or quadratic space were known. The tradeoff shows for the first time that we can simultaneously achieve subexponential time and subquadratic space. The boundary values are the exponential time with hardly any extra space required by the Lange-McKenzie-Tapp method and the $(\log 3)$th power time with square space required by the Bennett method. We also give the first general lower bound on the extra storage space required by general reversible simulation. This lower bound is optimal in that it is achieved by some reversible simulations.

1 Introduction

Computer power has roughly doubled every 18 months for the last half-century (Moore's law). This increase in power is due primarily to the continuing miniaturization of the elements of which computers are made, resulting in more and more elementary gates per unit area with higher and higher clock frequency, accompanied by less and less energy dissipation per elementary computing event. Roughly, a linear increase in clock speed is accompanied by a square increase in elements per unit area—so if all elements compute all of the time, then the dissipated energy per time unit rises cubicly (linear times square) in absence of energy decrease per elementary event. The continuing dramatic decrease in dissipated energy per elementary event is what has made Moore's law possible. But there is a foreseeable end to this: There is a minimum quantum of energy dissipation associated with elementary events. This puts a fundamental limit on how far we can go with miniaturazation, or does it?

Reversible Computation: R. Landauer [8] has demonstrated that it is only the 'logically irreversible' operations in a physical computer that necessarily dissipate energy by generating a corresponding amount of entropy for every

* All authors are partially supported by the EU fifth framework project QAIP, IST–1999–11234, the NoE QUIPROCONE IST–1999–29064, the ESF QiT Programmme, and the EU Fourth Framework BRA NeuroCOLT II Working Group EP 27150. Buhrman and Vitányi are also affiliated with the University of Amsterdam. Tromp is also affiliated with Bioinformatics Solutions, Waterloo, N2L 3G1 Ontario, Canada.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 1017–1027, 2001.

bit of information that gets irreversibly erased; the logically reversible operations can in principle be performed dissipation-free. Currently, computations are commonly irreversible, even though the physical devices that execute them are fundamentally reversible. At the basic level, however, matter is governed by classical mechanics and quantum mechanics, which are reversible. This contrast is only possible at the cost of efficiency loss by generating thermal entropy into the environment. With computational device technology rapidly approaching the elementary particle level it has been argued many times that this effect gains in significance to the extent that efficient operation (or operation at all) of future computers requires them to be reversible (for example, in [8,1,2,4,7,11,5]). The mismatch of computing organization and reality will express itself in friction: computers will dissipate a lot of heat unless their mode of operation becomes reversible, possibly quantum mechanical. Since 1940 the dissipated energy per bit operation in a computing device has—with remarkable regularity—decreased at the inverse rate of Moore's law [7] (making Moore's law possible). Extrapolation of current trends shows that the energy dissipation per binary logic operation needs to be reduced below kT (thermal noise) within 20 years. Here k is Boltzmann's constant and T the absolute temperature in degrees Kelvin, so that $kT \approx 3 \times 10^{-21}$ Joule at room temperature. Even at kT level, a future device containing 1 trillion (10^{12}) gates operating at 1 terahertz (10^{12}) switching all gates all of the time dissipates about 3000 watts. Consequently, in contemporary computer and chip architecture design the issue of power consumption has moved from a background worry to a major problem. For current research towards implementation of reversible computing on silicon see MIT's Pendulum Project and linked web pages (http://www.ai.mit.edu/∼cvieri/reversible.html). On a more futuristic note, quantum computing [15,14] is reversible. Despite its importance, theoretical advances in reversible computing are scarce and far between; all serious ones are listed in the references.

Related Work: Currently, almost no algorithms and other programs are designed according to reversible principles (and in fact, most tasks like computing Boolean functions are inherently irreversible). To write reversible programs by hand is unnatural and difficult. The natural way is to compile irreversible programs to reversible ones. This raises the question about efficiency of general reversible simulation of irreversible computation. Suppose the irreversible computation to be simulated uses T time and S space. A first efficient method was proposed by Bennett [3], but it is space hungry and uses [1] time $ST^{\log 3}$ and space $S \log T$. If T is maximal, that is, exponential in S, then the space use is S^2. This method can be modelled by a reversible pebble game. Reference [12] demonstrated that Bennett's method is optimal for reversible pebble games and that simulation space can be traded off against limited erasing. In [9] it was shown that using a method by Sipser [16] one can reversibly simulate using only $O(S)$ extra space but at the cost of using exponential time. In [6] the authors provide

[1] By judicious choosing of simulation parameters this method can be tweaked to run in $ST^{1+\epsilon}$ time for every $\epsilon > 0$ at the cost of introducing a multiplicative constant depending on $1/\epsilon$. The complexity analysis of [3] was completed in [10].

an oracle construction (essentially based on [12]) that separates reversible and irreversible space-time complexity classes.

Results: Previous results seem to suggest that a reversible simulation is stuck with either quadratic space use or exponential time use. This impression turns out to be false: [2]

Here we prove a tradeoff between time and space which has the exponential time simulation and the quadratic space simulation as extremes and for the first time gives a range of simulations using simultaneously subexponential ($2^{f(n)}$ is subexponential if $f(n) = o(n)$) time and subquadratic space. The idea is to use Bennett's pebbling game where the pebble steps are intervals of the simulated computation that are bridged by using the exponential simulation method. (It should be noted that embedding Bennett's pebbling game in the exponential method gives no gain, and neither does any other iteration of embeddings of simulation methods.) Careful analysis shows that the simulation using k pebbles takes $T' := S3^k 2^{O(T/2^k)}$ time and $S' = O(kS)$ space, and in some cases the upper bounds are tight. For $k = 0$ we have the exponential time simulation method and for $k = \log T$ we have Bennett's method. Interesting values arise for say

(a) $k = \log\log T$: $T' = S(\log T)^{\log 3} 2^{O(T/\log T)}$ and $S' = S\log\log T \leq S\log S$;

(b) $k = \sqrt{\log T}$: $S' = S\sqrt{\log T} \leq S\sqrt{S}$ and $T' = S3^{\sqrt{\log T}} 2^{O(T/2^{\sqrt{\log T}})}$.

(c) Let T, S, T', S' be as above. Eliminating the unknown k shows the tradeoff between simulation time T' and extra simulation space S': $T' = S3^{\frac{S'}{S}} 2^{O(T/2^{\frac{S'}{S}})}$.

(d) Let T, S, T', S' be as above and let the irreversible computation be halting and compute a function from inputs of n bits to outputs. For general reversible simulation by a reversible Turing machine using a binary tape alphabet and a single tape, $S' \geq n + \log T + O(1)$ and $T' \geq T$. This lower bound is optimal in the sense that it can be achieved by simulations at the cost of using time exponential in S.

Main open problem: The ultimate question is whether one can do better, and obtain improved upper and lower bounds on the tradeoff between time and space of reversible simulation, and in particular whether one can have almost linear time and almost linear space simultaneously.

2 Reversible Turing Machines

In the standard model of a Turing machine the elementary operations are rules in quadruple format (p, s, a, q) meaning that if the finite control is in state p and the machine scans tape symbol s, then the machine performs action a and subsequently the finite control enters state q. Such an action a consists of either printing a symbol s' in the tape square scanned, or moving the scanning head one tape square left or right.

[2] The work reported in this paper dates from 1998; Dieter van Melkebeek has drawn our attention to the unpublished [17] with similar, independent but later, research.

Quadruples are said to *overlap in domain* if they cause the machine in the same state and scanning the same symbol to perform different actions. A *deterministic Turing machine* is defined as a Turing machine with quadruples no two of which overlap in domain.

Now consider the special format (deterministic) Turing machines using quadruples of two types: *read/write* quadruples and *move* quadruples. A read/write quadruple (p, a, b, q) causes the machine in state p scanning tape symbol a to write symbol b and enter state q. A move quadruple $(p, *, \sigma, q)$ causes the machine in state p to move its tape head by $\sigma \in \{-1, +1\}$ squares and enter state q, oblivious to the particular symbol in the currently scanned tape square. (Here '-1' means 'one square left', and '$+1$' means 'one square right'.) Quadruples are said to *overlap in range* if they cause the machine to enter the same state and either both write the same symbol or (at least) one of them moves the head. Said differently, quadruples that enter the same state overlap in range unless they write different symbols. A *reversible Turing machine* is a deterministic Turing machine with quadruples no two of which overlap in range. A k-tape reversible Turing machine uses $(2k + 2)$ tuples which, for every tape separately, select a read/write or move on that tape. Moreover, any two tuples can be restricted to some single tape where they don't overlap in range.

To show that every partial recursive function can be computed by a reversible Turing machine one can proceed as follows [1]. Take the standard irreversible Turing machine computing that function. We modify it by adding an auxiliary storage tape called the 'history tape'. The quadruple rules are extended to 6-tuples to additionally manipulate the history tape. To be able to reversibly undo (retrace) the computation deterministically, the new 6-tuple rules have the effect that the machine keeps a record on the auxiliary history tape consisting of the sequence of quadruples executed on the original tape. Reversibly undoing a computation entails also erasing the record of its execution from the history tape. This notion of reversible computation means that only $1 : 1$ recursive functions can be computed. To reversibly simulate an irreversible computation from x to $f(x)$ one reversibly computes from input x to output $\langle x, f(x)\rangle$.

Reversible Turing machines or other reversible computers will require special reversible programs. One feature of such programs is that they should be executable when read from bottom to top as well as when read from top to bottom. Examples are the programs $F(\cdot)$ and $A(\cdot)$ in [12]. In general, writing reversible programs will be difficult. However, given a general reversible simulation of irreversible computation, one can simply write an oldfashioned irreversible program in an irreversible programming language, and subsequently simulate it reversibly. This leads to the following:

Definition 1. *An* irreversible-to-reversible compiler *receives an irreversible program as input and compiles it to a reversible program.*

Note that there is a decisive difference between reversible circuits and reversible special purpose computers [4] on the one hand, and reversible universal computers on the other hand [1,3]. While one can design a special-purpose reversible version for every particular irreversible circuit using reversible universal gates, such

a method does not yield an irreversible-to-reversible compiler that can execute any irreversible program on a fixed universal reversible computer architecture as we are interested in here.

3 Time Parsimonious Simulation

3.1 Background

We keep the discussion at an intuitive informal level; the cited references contain the formal details and rigorous constructions. An irreversible deterministic Turing machine has an infinite graph of *all* configurations where every configuration has outdegree at most one. In a reversible deterministic Turing machine every configuration also has indegree at most one. The problem of reversing an irreversible computation from its output is to revisit the input configurations starting from the output configuration by a process of reversibly traversing the graph.

The reversible Bennett strategy [3] essentially reversibly visits only the linear graph of configurations visited by the irreversible deterministic Turing machine in its computation from input to output, and no other configurations in the graph. It does so by a recursive procedure of establishing and undoing intermediate checkpoints that are kept simultanously in memory. It turns out that this can be done using limited time $T^{\log 3}$ and space $S \log T$.

3.2 Reversible Pebbling

Let G be a linear list of nodes $\{1, 2, \ldots, T_G\}$. We define a *pebble game* on G as follows. The game proceeds in a discrete sequence of steps of a single *player*. There are n pebbles which can be put on nodes of G. At any time the set of pebbles is divided in pebbles on nodes of G and the remaining pebbles which are called *free* pebbles. At every step either an existing free pebble can be put on a node of G (and is thus removed from the free pebble pool) or be removed from a node of G (and is added to the free pebble pool). Initially G is unpebbled and there is a pool of free pebbles. The game is played according to the following rule:

Reversible Pebble Rule: If node i is occupied by a pebble, then one may either place a free pebble on node $i + 1$ (if it was not occupied before), or remove the pebble from node $i + 1$.

We assume an extra initial node 0 permanently occupied by an extra, fixed pebble, so that node 1 may be (un)pebbled at will. This pebble game is inspired by the method of simulating irreversible Turing Machines on reversible ones in a space efficient manner. The placement of a pebble corresponds to checkpointing the next state of the irreversible computation, while the removal of a pebble corresponds to reversibly erasing a checkpoint. Our main interest is in determining the number of pebbles k needed to pebble a given node i.

The maximum number n of pebbles which are simultaneously on G at any one time in the game gives the space complexity nS of the simulation. If one deletes a pebble not following the above rules, then this means a block of bits of size S is erased irreversibly.

3.3 Algorithm

We describe the idea of Bennett's simulation [3]. This simulation is optimal [12] among all reversible pebble games. The total computation of T steps is broken into 2^k segments of length $m = T2^{-k}$. Every mth point of the computation is a node in the pebbling game; node i corresponding to im steps of computation.

For each pebble a section of tape is reserved long enough to store the whole configuration of the simulated machine. By enlarging the tape alphabet, each pebble will require space only $S + O(1)$.

Both the pebbling and unpebbling of a pebble t on some node, given that the previous node has a pebble s on it, will be achieved by a single reversible procedure bridge(s, t). This looks up the configuration at section s, simulates m steps of computation in a manner described in section 4, and exclusive-or's the result into section t. If t was a free pebble, meaning that its tape section is all zeroes, the result is that pebble t occupies the next node. If t already pebbled that node then it will be zeroed as a result.

The essence of Bennett's simulation is a recursive subdivision of a computation path into 2 halves, which are traversed in 3 stages; the first stage gets the midpoint pebbled, the second gets the endpoint pebbled, and the 3rd recovers the midpoint pebble. The following recursive procedure implements this scheme; Pebble(s, t, n) uses free pebbles $0, \ldots, n-1$ to compute the 2^nth node after the one pebbled by s, and exclusive-or's that node with pebble t (either putting t on the node or taking it off). Its correctness follows by straightforward induction. Note that it is its own reverse; executing it twice will produce no net change. The pebble parameters s and t are simply numbers in the range $-1, 0, 1, \ldots, k$. Pebble -1 is permanently on node 0, pebble k gets to pebble the final node, and pebble i, for $0 \leq i < k$ pebbles nodes that are odd multiples of 2^i. The entire simulation is carried out with a call pebble$(-1, k, k)$.

```
pebble(s, t, n)
{
  if (n = 0)
    bridge(s, t);
  fi (n = 0)
  if (n > 0)
  let r = n − 1
  pebble(s, r, n − 1);
  pebble(r, t, n − 1);
  pebble(s, r, n − 1)
  fi (n > 0)
}
```

As noted by Bennett, both branches and merges must be labeled with mutually exclusive conditions to ensure reversibility. Recursion can be easily implemented reversibly by introducing an extra stack tape, which will hold at most n stack frames of size $O(\log n)$ each, for a total of $O(n \log n)$.

This pebbling method is optimal in that no more than $2^{n+1} - 1$ steps can be bridged with n pebbles [12]. A call pebble(s, t, n) results in 3^n calls to bridge$(\cdot,\cdot)$. Bennett chose the number of pebbles large enough ($n = \Omega(\log T)$) so that m becomes small, on the order of the space S used by the simulated machine. In that case bridge(s, t) is easily implemented with the help of an additional *history* tape of size m which records the sequence of transitions. Instead, we allow an arbitrary choice of n and resort to the space efficient simulation of [9] to bridge the pebbled checkpoints.

4 Space Parsimonious Simulation

Lange, McKenzie and Tapp, [9], devised a reversible simulation, *LMT-simulation* for short, that doesn't use extra space, at the cost of using exponential time. Their main idea of reversibly simulating a machine without using more space is by reversibly cycling through the configuration tree of the machine (more precisely the connected component containing the input configuration). This configuration tree is a tree whose nodes are the machine configurations and where two nodes are connected by an edge if the machine moves in one step from one configuration to the other. We consider each edge to consist of two *half-edges*, each adjacent to one configuration.

The configuration tree can be traversed by alternating two permutations on half-edges: a swapping permutation which swaps the two half-edges constituting each edge, and a rotation permutation whose orbits are all the half-edges adjacent to one configuration. Both permutations can be implemented in a constant number of steps. For simplicity one assumes the simulated machine strictly alternates moving and writing transitions. To prevent the simulation from exceeding the available space S, each pebble section is marked with special left and right markers $\dagger, \ddagger$, which we assume the simulated machine not to cross. Since this only prevents crossings in the forward simulation, we furthermore, with the head on the left (right) marker, only consider previous moving transitions from the right (left).

5 The Tradeoff Simulation

To adapt the LMT simulation to our needs, we equip our simulating machine with one extra tape to hold the simulated configuration and another extra tape counting the difference between forward and backward steps simulated. $m = 2^n$ steps of computation can be bridged with a $\log m$ bits binary counter, incremented with each simulated forward step, and decremented with each simulated backward step— incurring an extra $O(\log m)$ factor slowdown in simulation speed. Having obtained the configuration m steps beyond that of pebble

s, it is exclusive-or'd into section t and then the LMT simulation is reversed to end up with a zero counter and a copy of section s, which is blanked by an exclusive-or from the original.

```
bridge(s, t)
{
  copy section s onto (blanked) simulation tape
  setup: goto enter;
  loop1: come from endloop1;
  simulate step with swap&rotate and adjust counter
  if (counter=0)
    rotate back;
    if (simulation tape = section s)
      enter: come from start;
    fi (simulation tape = section s)
  fi (counter=0)
  endloop1: if (counter!=m) goto loop1;
  exclusive-or simulation tape into section t
  if (counter!=m)
    loop2: come from endloop2;
  reverse-simulate step with anti-rotate&swap and adjust counter
  if (counter=0)
    rotate back;
    if (simulation tape = section s) goto exit;
  fi (counter=0)
  endloop2: goto loop2;
  exit: clear simulation tape using section s
}
```

5.1 Complexity Analysis

Let us analyze the time and space used by this simulation.

Theorem 1. *An irreversible computation using time T and space S can be simulated reversibly in time $T' = 3^k 2^{O(T/2^k)} S$ and space $S' = S(1 + O(k))$, where k is a parameter that can be chosen freely $0 \leq k \leq \log T$ to obtain the required tradeoff between reversible time T' and space S'.*

Proof. (Sketch) Every invocation of the bridge() procedure takes time $O(2^{O(m)}S)$. That is, every configuration has at most $O(1)$ predecessor configurations where it can have come from (constant number of states, constant alphabet size and choice of direction). Hence there are $\leq 2^{O(m)}$ configurations to be searched and about as many potential start configurations leading in m moves to the goal configuration, and every tape section comparison takes time $O(S)$. The pebbling game over 2^k nodes takes 3^k (un)pebbling steps each of which is an invocation of bridge(). Filling in $m = T/2^k$ gives the claimed time bound. Each of the $k + O(1)$ pebbles takes space $O(S)$, as does the simulation tape and the counter, giving the claimed total space. □

It is easy to verify that for some simulations the upper bound is tight. The boundary cases, $k = 0$ gives the LMT-simulation using exponential time and no extra space, and $k = \log T$ gives Bennett's simulation using at most square space and subquadratic time. Taking intermediate values of k we can choose to reduce time at the cost of an increase of space use and vice versa. In particular, special values $k = \log \log T$ and $k = \sqrt{T}$ give the results using simultaneously subexponential time and subquadratic space exhibited in the introduction. Eliminating k we obtain:

Corollary 1. *Let T, S, T', S' be as above. Then there is a reversible simulation that has the following tradeoff between simulation time T' and extra simulation space S':*

$$T' = S 3^{\frac{S'}{S}} 2^{O(T/2^{\frac{S'}{S}})}.$$

5.2 Local Irreversible Actions

Suppose we have an otherwise reversible computation containing local irreversible actions. Then we need to reversibly simulate only the subsequence of irreversible steps, leaving the connecting reversible computation segments unchanged. That is, an irreversiblity parsimonious computation is much cheaper to reversibly simulate than an irreversibility hungry one.

5.3 Reversible Simulation of Unknown Computing Time

In the previous analysis we have tacitly assumed that the reversible simulator knows in advance the number of steps T taken by the irreversible computation to be simulated. In this context one can distinguish on-line computations and off-line computations to be simulated. On-line computations are computations which interact with the outside environment and in principle keep running forever. An example is the operating system of a computer. Off-line computations are computations which compute a definite function from an input (argument) to an output (value). For example, given as input a positive integer number, compute as output all its prime factors. For every input such an algorithm will have a definite running time.

There is a well-known simple device to remove this dependency for batch computations without increasing the simulation time (and space) too much. Suppose we want to simulate a computation with unknown computation time T. Then we simulate t steps of the computation with t running through the sequence of values $2, 2^2, 2^3, \ldots$ For every value t takes on we reversibly simulate the first t steps of the irreversible computation. If $T > t$ then the computation is not finished at the end of this simulation. Subsequently we reversibly undo the computation until the initial state is reached again, set $t := 2t$ and reversibly simulate again. This way we continue until $t \geq T$ at which bound the computation finishes. The total time spent in this simulation is

$$T'' \leq 2 \sum_{i=1}^{\lceil \log T \rceil} S 3^{\frac{S'}{S}} 2^{O(2^{i-\frac{S'}{S}})} \leq 2T'.$$

6 Lower Bound on Reversible Simulation

It is not difficult to show a simple lower bound on the extra storage space required for general reversible simulation. We consider only irreversible computations that are halting computations performing a mapping from an input to an output. For convenience we assume that the Turing machine has a single binary work tape delemited by markers $\dagger, \ddagger$ that are placed S positions apart. Initially the binary input of length n is written left adjusted on the work tape. At the end of the computation the output is written left adjusted on the work tape. The markers are never moved. Such a machine clearly can perform every computation as long as S is large enough with respect to n. Assume that the reversible simulator is a similar model albeit reversible. The average number of steps in the computation is the uniform average over all equally likely inputs of n bits.

Theorem 2. *To generally simulate an irreversible halting computation of a Turing machine as above using storage space S and T steps on average, on inputs of length n, by a general reversible computation using S' storage space and T' steps on average, the reversible simulator Turing machine having q' states, requires trivially $T' \geq T$ and $S' \geq n + \log T - O(1)$ up to a logarithmic additive term.*

Proof. There are 2^n possible inputs to the irreversible computation, the computation on every input using on average T steps. A general simulation of this machine cannot use the semantics of the function being simulated but must simulate every step of the simulated machine. Hence $T' \geq T$. The simulator being reversible requires different configurations for every step of everyone of the simulated computations that is, at least $2^n T$ configurations. The simulating machine has not more than $q' 2^{S'} S'$ distinct configurations—$2^{S'}$ distinct values on the work tape, q' states, and S' head positions for the combination of input tape and work tape. Therefore, $q' 2^{S'} S' \geq 2^n T$. That is, $q' S' 2^{S'-n} \geq T$ which shows that $S' - n - \log S' \geq \log T - \log q'$. □

For example, consider irreversible computations that don't use extra space apart from the space to hold the input, that is, $S = n$. An example is the computation of $f(x) = 0$.

- If T is polynomial in n then $S' = n + \Omega(\log n)$.
- If T is exponential in n then $S' = n + \Omega(n)$.

Thus, in some cases the LMT-algorithm is required to use extra space if we deal with halting computations computing a function from input to output. In the final version of the paper [9] the authors have added that their simulation uses some extra space for counting (essentially $O(S)$) in case we require halting computations from input to output, matching the lower bound above for $S = n$ since their simulation uses on average T' steps exponential in S.

Optimality and Tradeoffs: The lower bound of Theorem 2 is optimal in the following sense. As one extreme, the LMT-algorithm of [9] discussed above uses $S' = n + \log T$ space for simulating irreversible computations of total functions on inputs of n bits, but at the cost of using $T' = \Omega(2^S)$ simulation time. As the other

extreme, Bennett's simple algorithm in [1] uses $T' = O(T)$ reversible simulation time, but at the cost of using $S' = \Omega(T)$ additional storage space. This implies that improvements in determining the complexity of reversible simulation must consider time-space tradeoffs.

References

1. C.H. Bennett. Logical reversibility of computation. *IBM J. Res. Develop.*, 17:525–532, 1973.
2. C.H. Bennett. The thermodynamics of computation—a review. *Int. J. Theoret. Phys.*, 21(1982), 905-940.
3. C.H. Bennett. Time-space tradeoffs for reversible computation. *SIAM J. Comput.*, 18(1989), 766-776.
4. E. Fredkin and T. Toffoli. Conservative logic. *Int. J. Theoret. Phys.*, 21(1982),219-253.
5. M. Frank, T. Knight, and N. Margolus, Reversibility in optimally scalable computer architectures, Manuscript, MIT-LCS, 1997 //`http://www.ai.mit.edu/~mpf/publications.html`.
6. M.P. Frank and M.J. Ammer, Separations of reversible and irreversible space-time complexity classes, Submitted. //`http://www.ai.mit.edu/~mpf/rc/memos/M06_oracle.html`.
7. R.W. Keyes, *IBM J. Res. Dev.*, 32(1988), 24-28.
8. R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Develop.*, 5:183–191, 1961.
9. K.J. Lange, P. McKenzie, and A. Tapp, Reversible space equals deterministic space, *J. Comput. System Sci.*, 60:2(2000), 354–367.
10. R.Y. Levine and A.T. Sherman, A note on Bennett's time-space tradeoff for reversible computation, *SIAM J. Comput.*, 19:4(1990), 673-677.
11. M. Li and P.M.B. Vitányi, Reversibility and adiabatic computation: trading time and space for energy, *Proc. Royal Society of London, Series A*, 452(1996), 769-789.
12. M. Li, J. Tromp, and P. Vitányi, Reversible simulation of irreversible computation. *Physica D*, 120(1998) 168-176.
13. K. Morita, A. Shirasaki, and Y. Gono, A 1-tape 2-symbol reversible Turing machine, *IEEE Trans. IEICE*, E72 (1989), 223–228.
14. M. Nielsen, I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.
15. P.W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26:5(1997), 1484–1509.
16. M. Sipser, Halting space-bounded computation, *Theoret. Comp. Sci.*, 10(1990), 335–338.
17. R. Williams, Space-Efficient Reversible Simulations, DIMACS REU report, July 2000. `http://dimacs.rutgers.edu/~ryanw/`

Finite-State Dimension

Jack J. Dai[1⋆], James I. Lathrop[2], Jack H. Lutz[2⋆], and Elvira Mayordomo[3⋆⋆]

[1] Department of Mathematics,
Iowa State University, Ames, IA 50011, USA
jdai@iastate.edu
[2] Department of Computer Science,
Iowa State University, Ames, IA 50011, USA,
jil@cs.iastate.edu, lutz@cs.iastate.edu
[3] Departamento de Informática e Ing. de Sistemas,
Universidad de Zaragoza, 50015 Zaragoza, SPAIN,
elvira@posta.unizar.es

Abstract. Classical Hausdorff dimension was recently effectivized using gales (betting strategies that generalize martingales), thereby endowing various complexity classes with dimension structure and also defining the constructive dimensions of individual binary (infinite) sequences. In this paper we use gales computed by multi-account finite-state gamblers to develop the finite-state dimensions of sets of binary sequences and individual binary sequences. Every rational sequence (binary expansion of a rational number) has finite-state dimension 0, but every rational number in $[0, 1]$ is the finite-state dimension of a sequence in the low-level complexity class AC_0. Our main theorem shows that the finite-state dimension of a sequence is precisely the infimum of all compression ratios achievable on the sequence by information-lossless finite-state compressors.

1 Introduction

Hausdorff dimension, best known as a powerful tool of fractal geometry, has been known for over fifty years to be closely related to information. For example, Eggleston [3] proved that in the space of all infinite binary sequences, if we let $\mathrm{FREQ}(\alpha)$ be the set of sequences in which 1 appears with asymptotic frequency α $(0 \leq \alpha \leq 1)$, then the Hausdorff dimension of $\mathrm{FREQ}(\alpha)$ is precisely $\mathcal{H}(\alpha)$, the binary entropy of α. More recent investigations of Ryabko [13,14,15], Staiger [17, 18], and Cai and Hartmanis [2] have explored relationships between Hausdorff dimension and Kolmogorov complexity (algorithmic information).

⋆ This research was supported in part by National Science Foundation Grants 9610461 and 9988483.

⋆⋆ This research was supported in part by Spanish Government MEC projects PB98-0937-C04-02 and TIC98-0973-C03-02. It was done while visiting Iowa State University.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 1028–1039, 2001.

Hausdorff dimension was originally defined topologically, using open covers by balls of diminishing radii [6,4]. Very recently, Lutz [11] proved a new characterization of Hausdorff dimension in terms of *gales*, which are betting strategies that generalize martingales. Lutz used this characterization to effectivize Hausdorff dimension, thereby defining dimension in complexity classes [11] and the constructive dimensions of individual sequences [12].

In this paper we extend the effectivization of dimension all the way to the level of finite-state computation. We define a *multi-account finite-state gambler* to be a finite-state gambler that maintains its capital in a portfolio of k separate accounts, so that capital in one account is shielded from losses in other accounts. (Finite-state gamblers with only one account have been investigated by Schnorr and Stimm [16] and Feder [5].) We use gales induced by multi-account finite-state gamblers to define the *finite-state dimension* $dim_{FS}(X)$ of each set $X \subseteq \mathbf{C}$, where $\mathbf{C}$ is the Cantor space, consisting of all infinite binary sequences. This definition is the natural finite-state effectivization of the gale characterization of classical Hausdorff dimension. In general, $dim_{FS}(X)$ is a real number satisfying

$$0 \leq dim_H(X) \leq dim_{FS}(X) \leq 1,$$

where $dim_H(X)$ is the Hausdorff dimension of X. Like Hausdorff dimension, finite-state dimension has the *stability* property that

$$dim_{FS}(X \cup Y) = \max\{dim_{FS}(X), dim_{FS}(Y)\}$$

for all $X, Y \subseteq \mathbf{C}$.

We show that finite-state dimension endows $\mathbf{Q}$, the set of all binary expansions of rational numbers in $[0, 1]$, with internal dimension structure. In particular, $\mathbf{Q}$ itself has finite-state dimension 1.

For an individual sequence $S \in \mathbf{C}$, we define the finite-state dimension of S to be $dim_{FS}(S) = dim_{FS}(\{S\})$. Each element of $\mathbf{Q}$ has finite-state dimension 0, while every sequence that is normal in the sense of Borel [1] has finite-state dimension 1. We show that every rational in $[0, 1]$ is the finite-state dimension of a sequence of very low complexity, namely, a sequence in the logspace-uniform version of the complexity class AC_0.

Our main theorem relates finite-state dimension to compressibility by information-lossless finite-state compressors, which were introduced by Huffman [7] and have been extensively investigated. (E.g., see [8] or [9].) Specifically, given such a compressor C and a sequence $S \in \mathbf{C}$, let $\rho_C(S)$ denote the limit infimum of all compression ratios achieved by C on prefixes of S, and let $\rho_{FS}(S)$ denote the infimum of all such $\rho_C(S)$. Our main theorem says that $dim_{FS}(S)$ is precisely $\rho_{FS}(S)$. Thus, with respect to finite-state computation, dimension and density of information are one and the same for individual sequences.

If $\rho_{LZ}(S)$ is the limit infimum of all compression ratios achieved by (any variant of) the Lempel-Ziv compression algorithm on prefixes of a sequence $S \in \mathbf{C}$, then it is well known that $\rho_{LZ}(S) \leq \rho_{FS}(S)$ [19]. Thus our main theorem implies that $\rho_{LZ}(S) \leq dim_{FS}(S)$. However, this inequality may be proper. For

example, Lathrop and Strauss [10] have shown that for every $\epsilon > 0$ there is a sequence S that is normal, whence $dim_{FS}(S) = 1$, but satisfies $\rho_{LZ}(S) < \epsilon$.

In the last part of the paper we discuss the role of multiple accounts in our model of finite-state gambling. Multiple accounts are necessary and sufficient for the associated class of gales to be closed under nonnegative, rational, linear combinations. However, we show that the restriction to single-account finite-state gamblers does not alter the finite-state dimension of any set of sequences. In our proof, the single-account gamblers have far more states than their multi-account counterparts, suggesting a possible tradeoff between accounts and states. It is an open question whether this tradeoff is real or merely a feature of our proof.

2 Preliminaries

If $S \in \mathbf{C}$, we write $S[i..j]$ for the finite string consisting of the i-th through the j-th bits of S. A sequence $S \in \mathbf{C}$ is *normal* [1] if each finite string x has asymptotic frequency $2^{-|x|}$ in S. That is, for each $x \in \{0,1\}^*$,

$$\lim_{n\to\infty} \frac{\left|\left\{i < n \;\middle|\; S[i\,..\,i+|x|-1] = x\right\}\right|}{n} = 2^{-|x|}.$$

NORM is the class of all normal sequences.

3 Finite-State Dimension

This section develops finite-state dimension and its fundamental properties. We first review the gale characterization of classical Hausdorff dimension, which motivates our development.

Definition. [11] Let $s \in [0,\infty)$

1. An *s-gale* is a function $d : \{0,1\}^* \to [0,\infty)$ that satisfies the condition
$$d(w) = 2^{-s}\left[d(w0) + d(w1)\right] \qquad (*)$$
for all $w \in \{0,1\}^*$.
2. A *martingale* is a 1-gale.

Intuitively, an s-gale is a strategy for betting on the successive bits of a sequence $S \in \mathbf{C}$. For each prefix w of S, $d(w)$ is the capital (amount of money) that d has after betting on the bits w of S. When betting on the next bit b of a prefix wb of S (assuming that b is equally likely to be 0 or 1), condition $(*)$ tells us that the expected value of $d(wb)$ – the capital that d expects to have after this bet – is $(d(w0) + d(w1))/2 = 2^{s-1}d(w)$. If $s = 1$, this expected value is exactly $d(w)$ – the capital that d has before the bet – so the payoffs are "fair." If $s < 1$, this expected value is less than $d(w)$, so the payoffs are "less than fair." Similarly, of $s > 1$, the payoffs are "more than fair."

The following generalization of the Kraft inequality and its corollaries will be useful.

Lemma 3.1. [11] Let $s \in [0, \infty)$. If d is an s-gale and $B \subseteq \{0,1\}^*$ is a prefix set, then for all $w \in \{0,1\}^*$,

$$\sum_{u \in B} 2^{-s|u|} d(wu) \leq d(w).$$

Corollary 3.2. [11] Let d be an s-gale, where $s \in [0, \infty)$. Then for all $w \in \{0,1\}^*, l \in \mathbb{N}$, and $0 < \alpha \in \mathbb{R}$, there are fewer than $\frac{2^l}{\alpha}$ strings $u \in \{0,1\}^l$ for which $d(wu) > \alpha 2^{(s-1)l} d(w)$.

Corollary 3.3. [11] If d is an s-gale, where $s \in [0, \infty)$, then for all $w, u \in \{0,1\}^*$,

$$d(wu) \leq 2^{s|u|} d(w).$$

Of course the objective of an s-gale is to win a lot of money.

Definition. Let d be an s-gale, where $s \in [0, \infty)$.

1. We say that d *succeeds* on a sequence $S \in \mathbf{C}$ if
$$\limsup_{n \to \infty} d(S[0..n-1]) = \infty.$$
2. The *success set* of d is
$$S^\infty[d] = \left\{ S \in \mathbf{C} \;\middle|\; d \text{ succeeds on } S \right\}.$$

Observation 3.4. Let $s, s' \in [0, \infty)$. For every s-gale d, the function $d' : \{0,1\}^* \to [0, \infty)$ defined by $d'(w) = 2^{(s'-s)|w|} d(w)$ is an s'-gale. If $s \leq s'$, then $S^\infty[d] \subseteq S^\infty[d']$.

Notation. For $X \subseteq \mathbf{C}, \mathcal{G}(X)$ is the set of all $s \in [0, \infty)$ such that there is an s-gale d for which $X \subseteq S^\infty[d]$.

It was shown in [11] that the following definition is equivalent to the classical definition of Hausdorff dimension in $\mathbf{C}$.

Definition. The *Hausdorff dimension* of a set $X \subseteq \mathbf{C}$ is

$$dim_H(X) = \inf \mathcal{G}(X).$$

In order to define finite-state dimension, we restrict attention to s-gales that are specified by finite-state devices. These devices place bets, which we require to be rational.

Definition. A *binary bet* is a rational number r such that $0 \leq r \leq 1$. We let $\mathbf{B}$ denote the set of all binary bets, i.e., $\mathbf{B} = \mathbb{Q} \cap [0, 1]$.

Intuitively, if a gambler whose current capital is $c \in [0, \infty)$ places a binary bet $r \in \mathbf{B}$ on a (perhaps unknown) bit $b \in \{0,1\}$, then the gambler is betting the fraction r of its capital that $b = 1$ and is betting the remainder of its capital

that $b = 0$. If the payoffs are fair, then after this bet the gambler's capital will be

$$2c[(1-b)(1-r)+br] = \begin{cases} 2rc & \text{if } b = 1, \\ 2(1-r)c & \text{if } b = 0. \end{cases}$$

We now introduce the model of finite-state gambling that is used to develop finite-state dimension. Intuitively, a finite-state gambler is a finite-state device that places k separate binary bets on each of the successive bits of its input sequence. These bets correspond to k separate accounts into which the gambler's capital is divided.

Definition. If k is a positive integer, then a *k-account finite-state gambler* (*k-account FSG*) is a 5-tuple

$$G = \left(Q, \delta, \vec{\beta}, q_0, \vec{c}_0\right),$$

where

- Q is a nonempty, finite set of *states*,
- $\delta : Q \times \{0,1\} \to Q$ is the *transition function*,
- $\vec{\beta} : Q \to \mathbf{B}^k$ is the *betting function*,
- $q_0 \in Q$ is the *initial state*, and
- $\vec{c}_0 = (c_{0,1}, \ldots, c_{0,k})$, the *initial capital vector*, is a sequence of nonnegative rational numbers.

A *finite state gambler (FSG)* is a k-account FSG for some positive integer k.

Note that we require $k > 0$. No-account gamblers are not worthy of discussion.

The case $k = 1$, where there is only one account, is the model of finite-state gambling that has been considered (in essentially equivalent form) by Schnorr and Stimm [16], Feder [5], and others. In this case we do not regard $\vec{c}_0$ as a vector, but simply as a nonnegative rational number c_0, which is the initial capital of G.

If $k > 1$, it is convenient to regard the betting function $\vec{\beta} : Q \to \mathbf{B}^k$ as a vector $\vec{\beta} = (\beta_1, \ldots, \beta_k)$ of component betting functions $\beta_i : Q \to \mathbf{B}$, so that

$$\vec{\beta}(q) = (\beta_1(q), \ldots, \beta_k(q))$$

for each $q \in Q$. If $k = 1$, we write β for $\vec{\beta}$.

As usual with finite-state transition structures, we extend δ to the transition function

$$\delta^* : Q \times \{0,1\}^* \to Q$$

defined by the recursion

$$\delta^*(q, \lambda) = q,$$
$$\delta^*(q, wb) = \delta(\delta^*(q, w), b)$$

for all $q \in Q$, $w \in \{0,1\}^*$, and $b \in \{0,1\}$; we write δ for δ^*; and we use the abbreviation $\delta(w) = \delta(q_0, w)$.

Intuitively, if a k-account FSG $G = (Q, \delta, \vec{\beta}, q_0, \vec{c}_0)$ is in state $q \in Q$ and its current capital vector is $\vec{c} = (c_1, \ldots, c_k) \in (\mathbb{Q} \cap [0,\infty))^k$, then for each of its accounts $i \in \{1, \ldots, k\}$, it places the binary bet $\beta_i(q) \in \mathbf{B}$. If the payoffs are fair, then after this bet G will be in state $\delta(q,b)$ and its i^{th} account will have capital

$$2c_i[(1-b)(1-\beta_i(q)) + b\beta_i(q)] = \begin{cases} 2\beta_i(q)c_i & \text{if } b = 1, \\ 2(1-\beta_i(q))c_i & \text{if } b = 0. \end{cases}$$

This suggests the following definition.

Definition. Let $G = (Q, \delta, \vec{\beta}, q_0, \vec{c}_0)$ be a k-account finite-state gambler.

1. For each $1 \le i \le k$, the i^{th} *martingale* of G is the function

$$d_{G,i} : \{0,1\}^* \to [0,\infty)$$

defined by the recursion

$$d_{G,i}(\lambda) = c_{0,i},$$
$$d_{G,i}(wb) = 2d_{G,i}(w)[(1-b)(1-\beta_i(\delta(w))) + b\beta_i(\delta(w))]$$

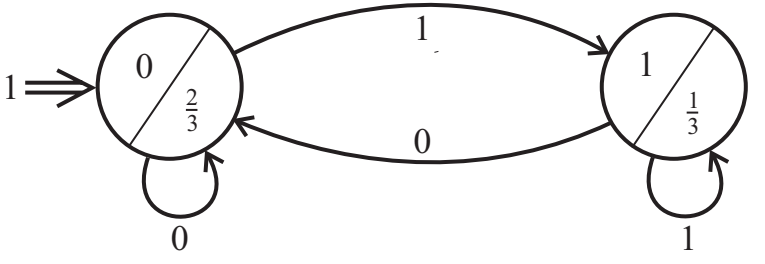
for all $w \in \{0,1\}^*$ and $b \in \{0,1\}$.

2. The *total martingale* (or simply the *martingale*) of G is the function

$$d_G = \sum_{i=1}^{k} d_{G,i}.$$

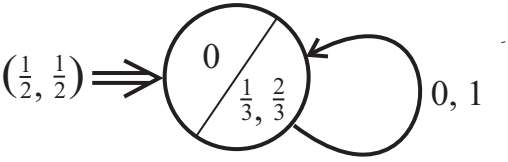
It is clear by inspection that $d_{G,1}, \ldots, d_{G,k}$, and d_G are all martingales for every k-account FSG G.

Example 3.5. The diagram



denotes the 1-account FSG $G = (Q, \delta, \beta, 0, 1)$, where $Q = \{0,1\}$, $\delta(0,0) = \delta(1,0) = 0$, $\delta(0,1) = \delta(1,1) = 1$, $\beta(0) = \frac{2}{3}$, and $\beta(1) = \frac{1}{3}$. It is easy to verify that $d_G(\lambda) = 1$, $d_G(1) = \frac{4}{3}$, $d_G(11) = \frac{8}{9}$, and $d_G(110) = \frac{32}{27}$.

Example 3.6. The diagram



denotes the 2-account FSG $G = (Q, \delta, \vec{\beta}, 0, (\frac{1}{2}, \frac{1}{2}))$, where $Q = \{0\}$, $\delta(0,0) = \delta(0,1) = 0$, $\beta_1(0) = \frac{1}{3}$, and $\beta_2(0) = \frac{2}{3}$. Although the two components of $\vec{\beta}$ make "opposite" bets, these do not "cancel" each other. For example, note that $d_G(00) = d_G(11) = \frac{10}{9} > 1 = d_G(0) = d_G(1)$. This is because the separation of accounts causes the effect of a component bet $\beta_i(q)$ to be proportional to the current capital in the i^{th} account.

Many of the k-account FSGs that we consider have the form $G = (Q, \delta, \vec{\beta}, q_0, \vec{c})$, where $\vec{c} = (\frac{1}{k}, \frac{1}{k}, \ldots, \frac{1}{k})$. In this case we omit $\vec{c}$ from the notation, referring simply to the k-account FSG $G = (Q, \delta, \vec{\beta}, q_0)$. Note that such a gambler always has initial capital $d_G(\lambda) = 1$.

Lemma 3.7. If G_1 is a k_1-account FSG with n_1 states and G_2 is a k_2-account FSG with n_2 states, then there is a $(k_1 + k_2)$-account FSG G with $n_1 n_2$ states such that $d_G = d_{G_1} + d_{G_2}$.

By Observation 3.4, an FSG G defines not only the martingale d_G, but also an s-gale for every s.

Definition. For $s \in [0, \infty)$, the *s-gale of* an FSG G is the function

$$d_G^{(s)} : \{0,1\}^* \to [0, \infty)$$

defined by

$$d_G^{(s)}(w) = 2^{(s-1)|w|} d_G(w)$$

for all $w \in \{0,1\}^*$.

In particular, note that $d_G^{(1)} = d_G$.

Definition.

1. For $s \in [0, \infty)$, a *finite-state s-gale* is an s-gale d for which there exists an FSG G such that $d_G^{(s)} = d$.
2. A *finite-state martingale* is a finite-state 1-gale.

We now use finite-state gales to define finite-state dimension.

Notation. For $X \subseteq \mathbf{C}$, $\mathcal{G}_{FS}(X)$ is the set of all $s \in [0, \infty)$ such that there is a finite-state s-gale d for which $X \subseteq S^\infty[d]$.

Observation 3.8. Let $X, Y \subseteq \mathbf{C}$ and $s, s' \in [0, \infty)$.

1. If $s' \geq s \in \mathcal{G}_{FS}(X)$, then $s' \in \mathcal{G}_{FS}(X)$.
2. $(1, \infty) \subseteq \mathcal{G}_{FS}(X) \subseteq (0, \infty)$.
3. $\mathcal{G}_{FS}(X) \subseteq \mathcal{G}(X)$.
4. If $X \subseteq Y$, then $\mathcal{G}_{FS}(Y) \subseteq \mathcal{G}_{FS}(X)$.

Recalling that the Hausdorff dimension of a set $X \subseteq \mathbf{C}$ is $dim_H(X) = \inf \mathcal{G}(X)$, it is natural to define the finite-state dimension as follows.

Definition. The *finite-state dimension* of a set $X \subseteq \mathbf{C}$ is

$$dim_{FS}(X) = \inf \mathcal{G}_{FS}(X).$$

Parts 1 and 2 of Observation 3.8 tell us that $\mathcal{G}_{FS}(X)$ is always of the form (s^*, ∞), where $0 \leq s^* \leq 1$, or of the form $[s^*, \infty)$, where $0 < s^* \leq 1$. In either case, the number s^* is the finite-state dimension of X.

Observation 3.8 has the following immediate consequences.

Observation 3.9. Let $X, Y \subseteq \mathbf{C}$.

1. $0 \leq dim_H(X) \leq dim_{FS}(X) \leq 1$.
2. If $X \subseteq Y$, then $dim_{FS}(X) \leq dim_{FS}(Y)$.

An important property of Hausdorff dimension is its *stability*, which is the term used by Falconer [4] for the fact that $dim_H(X \cup Y)$ is always the maximum of $dim_H(X)$ and $dim_H(Y)$. We now show that finite-state dimension has this property.

Theorem 3.10. For all $X, Y \subseteq \mathbf{C}$,

$$dim_{FS}(X \cup Y) = \max \{dim_{FS}(X), dim_{FS}(Y)\}.$$

Thus for all $X_1, \ldots, X_n \subseteq \mathbf{C}$,

$$dim_{FS}\left(\bigcup_{i=1}^{n} X_i\right) = \max_{1 \leq i \leq n} dim_{FS}(X_i).$$

We conclude this section with an easy technical lemma.

Definition. A 1-account FSG $G = (Q, \delta, \beta, q_0)$ is *nonvanishing* if $0 < \beta(q) < 1$ for all $q \in Q$.

Lemma 3.11. For every 1-account FSG G and every $\epsilon > 0$, there is a nonvanishing 1-account FSG G' such that for all $w \in \{0,1\}^*$, $d_{G'}(w) \geq 2^{-\epsilon|w|} d_G(w)$.

4 Rational Sequences

This brief section shows how to use finite-state dimension to define a natural notion of dimension in the set of all binary expansions of rational numbers.

Definition. Let $n \in \mathbb{Z}^+$ and $S \in \mathbf{C}$.

1. S is *eventually periodic* with *period* n, and we write $S \in \mathbf{Q}_n$, if there exist $x \in \{0,1\}^*$ and $y \in \{0,1\}^n$ such that for all $k \in \mathbb{N}$, $xy^k \sqsubseteq S$.
2. S is *eventually periodic*, and we write $S \in \mathbf{Q}$, if there exists $n \in \mathbb{Z}^+$ such that $S \in \mathbf{Q}_n$.

Note that for all $m, n \in \mathbb{Z}^+$, $\mathbf{Q}_n \subseteq \mathbf{Q}_{mn}$. Note also that $\mathbf{Q} = \cup_{n=1}^{\infty} \mathbf{Q}_n$ is precisely the set of all binary expansions of elements of $\mathbb{Q} \cap [0,1]$. For this reason, the elements of $\mathbf{Q}$ are also called *rational sequences*.

Lemma 4.1. For all $n \in \mathbb{Z}^+$, $dim_{FS}(\mathbf{Q}_n) = 0$.

In contrast with Lemma 4.1, and with the fact that every countable set of sequences has classical Hausdorff dimension 0, we have the following.

Theorem 4.2. $dim_{FS}(\mathbf{Q}) = 1$.

Our proof of Theorem 4.2 uses a result of Schnorr and Stimm [16] on 1-account FSGs. For each martingale d, let $X(d)$ be the set of all $S \in \mathbf{C}$ such that either

(i) d is *eventually constant* on S, i.e., $d(S\,[0..n]) = d(S\,[0..n-1])$ for all sufficiently large n, or
(ii) d *decays exponentially* on S, i.e., there exists $\alpha \in (0,1)$ such that $d(S\,[0..n-1]) < \alpha^n$ for all sufficiently large n.

Recall from section 2 that NORM is the set of all normal sequences.

Theorem 4.3. (Schnorr and Stimm [16]). If G is a 1-account FSG, then NORM $\subseteq X(d_G)$.

We now define dimension in the set of rational sequences.
Definition. For $X \subseteq \mathbf{C}$, the *dimension of* X *in* $\mathbf{Q}$ is

$$dim(X|\mathbf{Q}) = dim_{FS}(X \cap \mathbf{Q}).$$

This definition endows $\mathbf{Q}$ (and hence the set $\mathbb{Q}$ of rationals) with internal dimension structure. For example we have the following.

Observation 4.4. Let $X, Y \subseteq \mathbf{C}$.

1. $0 \leq dim(X|\mathbf{Q}) \leq 1$.
2. If $X \cap \mathbf{Q} \subseteq Y \cap \mathbf{Q}$, then $dim(X|\mathbf{Q}) \leq dim(Y|\mathbf{Q})$.
3. If $X \cap \mathbf{Q}$ is finite, then $dim(X|\mathbf{Q}) = 0$.
4. $dim(X \cup Y|\mathbf{Q}) = \max\{dim(X|\mathbf{Q}), dim(Y|\mathbf{Q})\}$.
5. $dim(\mathbf{Q}|\mathbf{Q}) = 1$.

5 Individual Sequences

It is natural to define the finite-state dimension of an individual sequence as follows.

Definition. The *finite-state dimension* of a sequence $S \in \mathbf{C}$ is

$$dim_{FS}(S) = dim_{FS}(\{S\}).$$

By Observation 4.4, every rational sequence has finite-state dimension 0. On the other hand, by Theorem 4.3, every normal sequence has finite-state dimension 1. The following theorem says that every rational number $r \in [0,1]$ is the finite-state dimension of a reasonably simple sequence.

Theorem 5.1. For every $r \in \mathbb{Q} \cap [0,1]$ there exists $S \in \mathrm{AC}_0$ such that $dim_{FS}(S) = r$.

6 Dimension and Compression

In this section we characterize the finite-state dimension of individual sequences in terms of finite-state compressibility. We first recall the definition of an information-lossless finite-state compressor. (This idea is due to Huffman [7]. Further exposition may be found in [8] or [9].)

Definition. A *finite-state compressor* (*FSC*) is a 4-tuple

$$C = (Q, \delta, \nu, q_0),$$

where

- Q is a nonempty, finite set of *states*,
- $\delta : Q \times \{0,1\} \to Q$ is the *transition function*,
- $\nu : Q \times \{0,1\} \to \{0,1\}^*$ is the *output function*, and
- $q_0 \in Q$ is the *initial state*.

For $q \in Q$ and $w \in \{0,1\}^*$, we define the *output* from state q on input w to be the string $\nu(q,w)$ defined by the recursion

$$\begin{aligned} \nu(q,\lambda) &= \lambda \\ \nu(q,wb) &= \nu(q,w)\nu(\delta(q,w),b) \end{aligned}$$

for all $w \in \{0,1\}^*$ and $b \in \{0,1\}$. We then define the *output* of C on input $w \in \{0,1\}^*$ to be the string

$$C(w) = \nu(q_0, w).$$

Definition. An FSC $C = (Q, \delta, \nu, q_0)$ is *information-lossless* (*IL*) if the function

$$\begin{aligned} \{0,1\}^* &\to \{0,1\}^* \times Q \\ w &\mapsto (C(w), \delta(w)) \end{aligned}$$

is one-to-one. An *information-lossless finite-state compressor* (*ILFSC*) is an FSC that is IL.

That is, an ILFSC is an FSC whose input can be reconstructed from the output and final state reached on that input.

Intuitively, an FSC C compresses a string w if $|C(w)|$ is significantly less than $|w|$. Of course, if C is IL, then not all strings can be compressed. Our interest here is in the degree (if any) to which the prefixes of a given sequence $S \in \mathbf{C}$ can be compressed by an ILFSC.

Definition. If C is an FSC and $S \in \mathbf{C}$, then the *compression ratio* of C on S is

$$\rho_C(S) = \liminf_{n\to\infty} \frac{|C(S[0\,..\,n-1])|}{n}.$$

Definition. The *finite-state compression ratio* of a sequence $S \in \mathbf{C}$ is

$$\rho_{FS}(S) = \inf\{\rho_C(S) | C \text{ is an ILFSC}\}.$$

The following theorem says that finite-state dimension and finite-state compressibility are one and the same.

Theorem 6.1. For all $S \in \mathbf{C}$,

$$dim_{FS}(S) = \rho_{FS}(S).$$

7 Accounts versus States

The multi-account model of finite-state gambling has the useful closure property that for every s, every nonnegative, rational, linear combination of finite-state s-gales is a finite-state s-gale. The single-account model of finite-state gambling does not enjoy this closure property. In fact, there is no 1-account FSG whose martingale is that of the 2-account FSG of Example 3.6. Nevertheless, the following theorem shows that finite-state dimension can equivalently be defined by the single-account model, provided that we are willing to accept a large increase in the number of states.

Theorem 7.1. For each n-state, k-account FSG G and each $\epsilon \in (0,1)$, if we let $m = \frac{\lceil \log k \rceil}{\epsilon}$ and $N = n(2^m - 1)$, then there is an N-state, 1-account FSG G' such that for all $s \in [0,1]$,

$$S^\infty[d_G^{(s)}] \subseteq S^\infty[d_{G'}^{(s+\epsilon)}].$$

At the time of this writing, we do not know whether such a large number of states is necessary.

Acknowledgment. We thank Giora Slutzki for useful discussions.

References

1. É. Borel. Sur les probabilités dénombrables et leurs applications arithmétiques. *Rend. Circ. Mat. Palermo*, 27:247–271, 1909.
2. J. Cai and J. Hartmanis. On Hausdorff and topological dimensions of the Kolmogorov complexity of the real line. *Journal of Computer and Systems Sciences*, 49:605–619, 1994.
3. H.G. Eggleston. The fractional dimension of a set defined by decimal properties. *Quarterly Journal of Mathematics*, Oxford Series 20:31–36, 1949.
4. K. Falconer. *The Geometry of Fractal Sets*. Cambridge University Press, 1985.
5. M. Feder. Gambling using a finite state machine. *IEEE Transactions on Information Theory*, 37:1459–1461, 1991.
6. F. Hausdorff. Dimension und äusseres Mass. *Math. Ann.*, 79:157–179, 1919.
7. D. A. Huffman. Canonical forms for information-lossless finite-state logical machines. *IRE Trans. Circuit Theory CT-6 (Special Supplement)*, pages 41–59, 1959. Also avalaible in E.F. Moore (ed.), Sequential Machine: Selected Papers, Addison-Wesley, 1964, pages 866-871.
8. Z. Kohavi. *Switching and Finite Automata Theory (Second Edition)*. McGraw-Hill, 1978.
9. A. A. Kurmit. *Information-Lossless Automata of Finite Order*. Wiley, 1974.
10. Lathrop and Strauss. A universal upper bound on the performance of the Lempel-Ziv algorithm on maliciously-constructed data. In *SEQS: Sequences '97*, 1997.
11. J. H. Lutz. Dimension in complexity classes. In *Proceedings of the Fifteenth Annual IEEE Conference on Computational Complexity*, pages 158–169. IEEE Computer Society Press, 2000.
12. J. H. Lutz. Gales and the constructive dimension of individual sequences. In *Proceedings of the 27th Colloquium on Automata, Languages and Programming*, pages 902–913. Springer Lecture Notes in Computer Science, 2000.
13. B. Ya. Ryabko. Noiseless coding of combinatorial sources. *Problems of Information Transmission*, 22:170–179, 1986.
14. B. Ya. Ryabko. Algorithmic approach to the prediction problem. *Problems of Information Transmission*, 29:186–193, 1993.
15. B. Ya. Ryabko. The complexity and effectiveness of prediction problems. *Journal of Complexity*, 10:281–295, 1994.
16. C. P. Schnorr and H. Stimm. Endliche automaten und zufallsfolgen. *Acta Informatica*, 1:345–359, 1972.
17. L. Staiger. Kolmogorov complexity and Hausdorff dimension. *Information and Computation*, 102:159–194, 1993.
18. L. Staiger. A tight upper bound on Kolmogorov complexity and uniformly optimal prediction. *Theory of Computing Systems*, 31:215–229, 1998.
19. J. Ziv. Coding theorem for individual sequences. *IEEE Transactions on Information Theory*, 24:405–412, 1978.

The Complexity of Computing the Size of an Interval*

Lane A. Hemaspaandra[1], Sven Kosub[2], and Klaus W. Wagner[2]

[1] Department of Computer Science, University of Rochester,
Rochester, NY 14627, USA
lane@cs.rochester.edu
[2] Theoretische Informatik, Julius-Maximilians-Universität Würzburg
Am Hubland, D-97074 Würzburg, Germany
{kosub,wagner}@informatik.uni-wuerzburg.de

Abstract. We study the complexity of counting the number of elements in intervals of feasible partial orders. Depending on the properties that partial orders may have, such counting functions have different complexities. If we consider total, polynomial-time decidable orders then we obtain exactly the #P functions. We show that the interval size functions for polynomial-time adjacency checkable orders are tightly related to the class FPSPACE(poly): Every FPSPACE(poly) function equals a polynomial-time function subtracted from such an interval size function. We study the function #DIV that counts the nontrivial divisors of natural numbers, and we show that #DIV is the interval size function of a polynomial-time decidable partial order with polynomial-time adjacency checks if and only if primality is in polynomial time.

1 Introduction

The class NP, widely believed to contain computationally intractable problems, captures the complexity of determining for a given problem instance whether at least one suitable affirmative solution exists within an exponentially large set of (polynomial-sized) potential solutions. It is seemingly much harder to count all affirmative solutions in such solution sets. The corresponding *counting functions* constitute Valiant's famous counting class #P [23]. In the theory of counting functions, which is devoted to the study of counting versions of decision problems, most classes considered try to capture the pure phenomenon of counting, and in doing so they obscure other factors, e.g., orders on solutions sets.

Natural counting problems in #P, of course, sometimes have strong relationships between solutions. As an example, consider the counting function #DIV which counts for each natural number the number of its nontrivial divisors. Clearly, #DIV is in #P since division can be done in polynomial time. A suitable structure in the set of solutions is the partial order $\leq_|$ of divisibility (that

* Supported in part by grants NSF-INT-9815095/DAAD-315-PPP-gü-ab and NSF-CCR-9322513. Work done in part while the first author was visiting Julius-Maximilians-Universität Würzburg.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 1040–1051, 2001.

is, $n \leq_| m$ iff n divides m). Obviously, $\#\mathrm{DIV}(m) = \|\{k \mid 1 <_| k <_| m\}\|$, i.e., $\#\mathrm{DIV}(m)$ counts the number of elements in the open interval $(1, m)$ in the partial order $\leq_|$ of natural numbers.

Is #DIV an exceptional case among #P functions that has such an interval size characterization? Interestingly, "no" is the answer. It turns out that a function f is in #P if and only if it is an interval size function of a P-decidable partial p-order. The latter means that there exist a *partial p-order* $\leq_A$ (i.e., a partial order such that, for some polynomial p and all x, y, it holds that $x \leq_A y$ implies $|x| \leq p(|y|)$) which is P-*decidable* (i.e., $x \leq_A y$ is decidable in polynomial time) and polynomial-time computable functions b and t such that $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$.

However, knowing that a partial p-order is polynomial-time decidable does not give us as much information as sometimes is needed. For example, the polynomial-time decidability of a p-order seemingly does not ensure that it has *efficient adjacency checks*, i.e., that there is a polynomial-time algorithm checking whether two elements are adjacent in this partial p-order. Indeed, if every P-decidable partial p-order has efficient adjacency checks then P = NP (and vice versa). Hence having additionally efficient adjacency checks seems to be a restriction. Denoting by $\mathrm{IF_p}$ (respectively, $\mathrm{IF_t}$) the class of interval size functions of P-decidable partial p-orders (respectively, total p-orders) with efficient adjacency checks, we obtain $\mathrm{IF_t} \subseteq \mathrm{IF_p} \subseteq \#\mathrm{P}$. On the one hand, we prove that $\mathrm{IF_t}\text{-}\mathrm{FP} = \mathrm{IF_p}\text{-}\mathrm{FP} = \#\mathrm{P}\text{-}\mathrm{FP}$, and thus these classes do not seem to be "very different." On the other hand, $\mathrm{IF_p} = \#\mathrm{P}$ is equivalent to P = NP, and hence it is very unlikely that $\mathrm{IF_p}$ and #P coincide. Further, we discover relationships between the classes $\mathrm{IF_t}$, FP, and $\mathrm{UPSV_t}$, as well as consequences of $\mathrm{IF_t} = \mathrm{IF_p}$.

We already mentioned that it is unlikely that every P-decidable partial p-order has efficient adjacency checks. What about the converse? This also is not likely; if every partial p-order with efficient adjacency checks is P-decidable then P = PSPACE (and vice versa). Hence, in the presence of efficient adjacency checks, removing the P-decidability requirement seems to be a relaxation. Denoting by $\mathrm{IF_p^*}$ (respectively, $\mathrm{IF_t^*}$) the class of interval size functions of partial p-orders (respectively, total p-orders) with efficient adjacency checks, we obtain $\mathrm{IF_p} \subseteq \mathrm{IF_p^*}$ and $\mathrm{IF_t} \subseteq \mathrm{IF_t^*}$. We prove that $\mathrm{IF_p^*}$ is remarkably powerful: $\mathrm{IF_p^*}\text{-}\mathrm{FP} = \mathrm{FPSPACE(poly)}\text{-}\mathrm{FP}$, which brings $\mathrm{IF_p^*}$ seemingly close to FPSPACE(poly), the class of polynomially length-bounded, polynomial-space computable functions. However, if these classes would really coincide then UP = PSPACE. We clarify further relations among such classes and also with respect to other function classes, in order to understand the power of interval computing.

Finally we prove that the counting function #DIV is in $\mathrm{IF_p}$ if and only if primality is in P.

Using order-theoretic notions to approach complexity issues is not new, and this appears in the literature in several settings (e.g., [7,6,24,10,12]) that, however, differ in intent from our approach. The notions from these studies focus

on one specific ordering, namely the lexicographical ordering. However, for our purposes it is essential to consider more general feasible orderings (see [15,11]).

Due to the page limit we omit some of the proofs here and defer them to the journal version.

2 Preliminaries

For our finite alphabet $\Sigma = \{0, 1\}$, let Σ^* be the set of all finite words (strings) built with letters from Σ. Let ε denote the empty word. The length of a word $x \in \Sigma^*$ is denoted by $|x|$. The complement of a set $L \subseteq \Sigma^*$ is denoted by $\overline{L}$, i.e., $\overline{L} = \Sigma^* \setminus L$. For any class $\mathcal{K}$ of subsets of Σ^*, let co$\mathcal{K}$ be the class $\{\overline{L} \mid L \in \mathcal{K}\}$. The cardinality of a set L is denoted by $\|L\|$. The characteristic function of a set $L \subseteq \Sigma^*$ is denoted by c_L, i.e., $\mathrm{c}_L(x) = 1 \Leftrightarrow x \in L$ and $\mathrm{c}_L(x) = 0 \Leftrightarrow x \notin L$, for all $x \in \Sigma^*$. Let $\mathbb{N} = \{0, 1, 2, \dots\}$.

For the basic notions of complexity theory such as P, NP, PSPACE, and so on see any standard text, e.g., [1,19]. The computation model we use is the standard nondeterministic Turing machine.

We review the definitions of some complexity classes of functions, already existing in the literature, that we will use in this paper.

- FP denotes the class of all polynomial-time computable, total functions from Σ^* to $\mathbb{N}$. We will at times use FP to denote the class of all polynomial-time computable, total functions from Σ^* to Σ^*; via the natural, efficient bijection between $\mathbb{N}$ and Σ^*, these are in essence the same.
- [13] FPSPACE(poly) is the class of all polynomial-space computable, total functions with polynomially length-bounded outputs.
- [23] #P is the class of all total functions f for which there exists a nondeterministic polynomial-time Turing machine M such that $f(x)$ is the number of accepting computations of M on input x. Equivalently, #P is the class of all total functions f for which there exist a set $B \in \mathrm{P}$ and a polynomial p such that $f(x) = \|\{z \mid |z| = p(|x|) \wedge (x, z) \in B\}\|$
- [8,12] $\mathrm{UPSV_t}$ is the class of all total functions f for which there exists a nondeterministic polynomial-time Turing machine M which always produces on exactly one computation path an output, and this output is $f(x)$.

For function classes $\mathcal{F}$ and $\mathcal{G}$, let $\mathcal{F}\text{-}\mathcal{G}$ denote the class of all functions h for which there exist functions $f \in \mathcal{F}$ and $g \in \mathcal{G}$ such that $h(x) = f(x) - g(x)$ for all $x \in \Sigma^*$. For a class $\mathcal{K}$ of sets, let $\mathrm{FP}^{\mathcal{K}}$ ($\mathrm{P}^{\mathcal{K}}$, resp.) be the class of functions (sets resp.) that can be computed in polynomial time with an oracle from $\mathcal{K}$.

Next we review the definitions of some complexity classes of sets, already existing in the literature, that we will use in this paper.

- [22] UP is the class of all sets L such that $\mathrm{c}_L \in \#\mathrm{P}$.
- [3,14] NP is the class of all sets L for which there exists a function $f \in \#\mathrm{P}$ such that $x \in L \Leftrightarrow f(x) > 0$ for all $x \in \Sigma^*$.
- [20,5] PP is the class of all sets L for which there exist functions $f \in \#\mathrm{P}$ and $g \in \mathrm{FP}$ such that $x \in L \Leftrightarrow f(x) \geq g(x)$ for all $x \in \Sigma^*$.

– [18,4] SPP is the class of all sets L such that $c_L \in \#\mathrm{P} - \mathrm{FP}$.
– [16,21] $\mathrm{PH} = \mathrm{NP} \cup \mathrm{NP}^{\mathrm{NP}} \cup \mathrm{NP}^{\mathrm{NP}^{\mathrm{NP}}} \cup \ldots$.

The following results are well-known or easy to see.

Proposition 1. *1.* $\mathrm{FP} \subseteq \mathrm{UPSV_t} = \mathrm{FP}^{\mathrm{UP} \cap \mathrm{coUP}} \subseteq \#\mathrm{P} \subseteq \mathrm{FPSPACE(poly)}$.
2. $\mathrm{P} \subseteq \mathrm{UP} \subseteq \mathrm{NP} \subseteq \mathrm{PH} \subseteq \mathrm{PSPACE}$.
3. $\mathrm{NP} \cup \mathrm{SPP} \subseteq \mathrm{PP}$.

Finally, we will use a complexity-theoretic operator of Hempel and Wechsung [9], which maps function classes to set classes: For a function class $\mathcal{F}$, let $\exists \cdot \mathcal{F}$ be the class of all sets L for which there exists a function $f \in \mathcal{F}$ such that $x \in L \Leftrightarrow f(x) > 0$ for all $x \in \Sigma^*$.

The following statements are immediate or easy to see.

Proposition 2. *1.* $\exists \cdot \mathrm{FP} = \mathrm{P}$.
2. $\exists \cdot \mathrm{UPSV_t} = \exists \cdot (\mathrm{UPSV_t} - \mathrm{FP}) = \exists \cdot (\mathrm{UPSV_t} - \mathrm{UPSV_t}) = \mathrm{UP} \cap \mathrm{coUP}$.
3. $\exists \cdot \#\mathrm{P} = \mathrm{NP}$.
4. $\exists \cdot (\#\mathrm{P} - \mathrm{FP}) = \mathrm{PP}$.
5. $\exists \cdot \mathrm{FPSPACE(poly)} = \mathrm{PSPACE}$.

3 Orders with Feasibility Constraints

In this section we introduce the order-theoretic notions that we need, and their polynomial-time versions (see also [11]).

A binary relation $A \subseteq \Sigma^* \times \Sigma^*$ is a *partial order* if it is reflexive, transitive, and antisymmetric. A partial order A is a *partial p-order* if there exists a polynomial p such that $|x| \leq p(|y|)$ for all $(x, y) \in A$. If A additionally satisfies $(x, y) \in A$ or $(y, x) \in A$ for all $x, y \in \Sigma^*$, then A is said to be a *total p-order*. Note that in a partial p-order for every string y there exist only exponentially (in the length of y) many strings that are less than y with respect to the order considered. Hence the interval size functions defined below are exponentially value-bounded relative to the input length in the same way as usually functions in polynomial-time settings, such as functions from FP and #P.

For any partial p-order A we make the following notational conventions. We write $x \leq_A y$ if $(x, y) \in A$, we write $x <_A y$ if $x \leq_A y$ and $x \neq y$, and we write $x \prec_A y$ if $x <_A y$ and there is no z with $x <_A z <_A y$. If $x \prec_A y$ we say that x *precedes* y or, equivalently, y *succeeds* x. Finally, we set $A_\prec =_{\mathrm{def}} \{(x, y) \mid x \prec_A y\}$.

Feasibility restrictions on orders are essential in our further studies. A P-*decidable partial p-order* is simply a partial p-order A that belongs to P. A partial p-order A has *efficient adjacency checks* if and only if the set $A_\prec$ is in P.

The two feasibility constraints discussed above can be expected to be incomparable. However, there are complexity-theoretic connections between both features.

Proposition 3. *Let A be a partial p-order.*

1. If $A \in \mathrm{P}$ then $A_\prec \in \mathrm{coNP}$.
2. If $A_\prec \in \mathrm{P}$ then $A \in \mathrm{PSPACE}$.

Corollary 4. *1. If* $\mathrm{P} = \mathrm{NP}$ *then all* P*-decidable partial p-orders have efficient adjacency checks.*
2. If $\mathrm{P} = \mathrm{PSPACE}$ *then all partial p-orders with efficient adjacency checks are* P*-decidable.*

In what follows we will see that the converse implications of the preceding corollary are also valid.

4 Orders without Efficient Adjacency Checks

In this section we will see how to use partial and total p-orders to capture function classes. In particular, we are interested in characterizing #P in terms of feasible p-orders. However, if we omit all feasibility restrictions on p-orders then, not surprisingly, all polynomially length-bounded functions can be characterized in the same way as intended for #P.

Theorem 5. *1. For any function* f *the following statements are equivalent.*
(1) $f \in \#\mathrm{P}$*.*
(2) There exist a partial p-order $A \in \mathrm{P}$ *and functions* $b, t \in \mathrm{FP}$ *such that* $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$ *for all* $x \in \Sigma^*$*.*
(3) There exist a total p-order $A \in \mathrm{P}$ *and functions* $b, t \in \mathrm{FP}$ *such that* $b(x) \leq_A t(x)$ *and* $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$ *for all* $x \in \Sigma^*$*.*
2. For any function f *the following statements are equivalent.*
(1) f *is polynomially length-bounded.*
(2) There exist a partial p-order A *and functions* $b, t \in \mathrm{FP}$ *such that* $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$ *for all* $x \in \Sigma^*$*.*
(3) There exist a total p-order A *and functions* $b, t \in \mathrm{FP}$ *such that* $b(x) \leq_A t(x)$ *and* $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$ *for all* $x \in \Sigma^*$*.*

Proof. The implications (3) $\Rightarrow$ (2) $\Rightarrow$ (1) are obvious; we prove (1) $\Rightarrow$ (3).

For every polynomially length-bounded function $f : \Sigma^* \to \mathbb{N}$ there exist a set $B \subseteq \Sigma^* \times \Sigma^*$ and a strictly monotonic polynomial p such that $f(x) = \|\{z \mid |z| = p(|x|) \wedge (x, z) \in B\}\|$. If, in addition, $f \in \#\mathrm{P}$ then B can be chosen to be from P. Without loss of generality assume that $(x, 0^{p(|x|)}), (x, 1^{p(|x|)}) \notin B$ for all x.

We construct a total p-order A on Σ^* as follows. Generally, A will coincide with the lexicographical order on Σ^* except, for every x, the interval between $x0^{p(|x|)}$ and $x1^{p(|x|)}$, which will be is ordered differently in the following way:

- first $x1^{p(|x|)}$,
- then the elements of $\{xz \mid |z| = p(|x|) \wedge (x, z) \in B\}$ in lexicographical order, and
- then the elements of $\{xz \mid |z| = p(|x|) \wedge (x, z) \notin B \wedge z \neq 1^{p(|x|)}\}$ in lexicographical order.

Obviously, $f(x) = \|\{w \mid x1^{p(|x|)} <_A w <_A x0^{p(|x|)}\}\|$. If, in addition, $B \in \mathrm{P}$ then also $A \in \mathrm{P}$. □

5 Polynomial-Time Orders with Efficient Adjacency Checks

From Theorem 5 in the preceding section we know that counting the size of intervals of (partial) p-orders $A \in \mathrm{P}$ just characterizes the functions of #P. The situation changes if we additionally assume efficient adjacency checks. The class $\mathrm{IF_p}$ ($\mathrm{IF_t}$, resp.) is defined as the class of all functions $f : \Sigma^* \to \mathbb{N}$ for which there exist a partial (total, resp.) p-order $A \in \mathrm{P}$ having efficient adjacency checks, and functions $b, t \in \mathrm{FP}$, such that $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$ for every $x \in \Sigma^*$. The following theorem places the classes $\mathrm{IF_t}$ and $\mathrm{IF_p}$ between well-known complexity classes.

Theorem 6. $\mathrm{FP} \subseteq \mathrm{IF_t} \subseteq \mathrm{IF_p} \subseteq \#\mathrm{P}$.

What can we say about the inclusions in Theorem 6? We start by showing that $\mathrm{IF_p}$ and #P coincide on Nonzero $=_{\mathrm{def}} \{f \mid f(x) > 0 \text{ for all } x\}$.

Theorem 7. $\mathrm{IF_p} \cap \mathrm{Nonzero} = \#\mathrm{P} \cap \mathrm{Nonzero}$.

Proof. Since $\mathrm{IF_p} \subseteq \#\mathrm{P}$, all that remains is to prove the inclusion "$\supseteq$."

For $f : \Sigma^* \to \mathbb{N}$ from #P such that $f(x) > 0$ for all x, there exists a set $B \subseteq \Sigma^* \times \Sigma^*$ from P and a strictly monotonic polynomial p such that $f(x) = \|\{z \mid |z| = p(|x|) \wedge (x, z) \in B\}\|$.

We construct a partial p-order A on Σ^* as follows. Generally, A coincides with the lexicographical order on Σ^* except, for every x, the interval between $x0^{p(|x|)}00$ and $x0^{p(|x|)}11$, which will be partially ordered in the following way:

- $x0^{p(|x|)}00 <_A x0^{p(|x|)}01 <_A x0^{p(|x|)}11$,
- the elements from $\{xz10 \mid |z| = p(|x|) \wedge (x, z) \in B\}$ are pairwise incomparable and all are between $x0^{p(|x|)}01$ and $x0^{p(|x|)}11$,
- the elements from $\{xz10 \mid |z| = p(|x|) \wedge (x, z) \notin B\} \cup \{xz\sigma \mid |z| = p(|x|) \wedge z \neq 0^{p(|x|)} \wedge \sigma \in \{00, 01, 11\}\}$ are pairwise incomparable and all are between $x0^{p(|x|)}00$ and $x0^{p(|x|)}01$.

Obviously, A is P-decidable, has efficient adjacency checks, and we obtain $\|\{w \mid x0^{p(|x|)}01 <_A w <_A x0^{p(|x|)}11\}\| = f(x)$. □

In what follows, we write 1 for the function class consisting of just the constant function $f(x) = 1$, and we write $\mathcal{O}(1)$ for the function class that contains exactly the functions $f(x) = 0$, $f(x) = 1$, $f(x) = 2$, ...

Corollary 8. *1.* $\#\mathrm{P} \subseteq \mathrm{IF_p} \cdot 1$.
2. $\#\mathrm{P} \cdot \mathcal{O}(1) = \mathrm{IF_p} \cdot \mathcal{O}(1)$.

From Theorem 6 and Corollary 8 we can conclude that $\mathrm{IF_p} \subseteq \mathrm{IF_p} \cdot 1$ which is equivalent to the fact that $\mathrm{IF_p}$ is closed under increment, i.e., for every $f \in \mathrm{IF_p}$ the function f' is also in $\mathrm{IF_p}$, where $f'(x) =_{\mathrm{def}} f(x)+1$ for all x.

Corollary 9. *The class* $\mathrm{IF_p}$ *is closed under increment.*

Corollary 8 is nicely complemented by the following theorem.

Theorem 10. *1.* $\#\mathrm{P} \subseteq \mathrm{IF_t} \cdot \mathrm{FP}$.
2. $\mathrm{IF_t} \cdot \mathrm{FP} = \mathrm{IF_p} \cdot \mathrm{FP} = \#\mathrm{P} \cdot \mathrm{FP}$.

Corollary 11. $\mathrm{FP}^{\mathrm{IF_t}} = \mathrm{FP}^{\mathrm{IF_p}} = \mathrm{FP}^{\#\mathrm{P}}$.

The previous results indicate that the computational power of $\mathrm{IF_p}$ and $\mathrm{IF_t}$ is not very much lower than that of #P. However, Theorem 13 shows that these classes cannot coincide unless P = NP. We start with a lemma on the application of the $\exists$ operator to $\mathrm{IF_p}$ and $\mathrm{IF_t}$. Comparing Lemma 12 with Theorem 7 and taking into account $\exists \cdot \#\mathrm{P} = \mathrm{NP}$, it turns out that it is just the possibility of $f(x) = 0$ that makes the classes #P and $\mathrm{IF_p}$ different.

Lemma 12. $\exists \cdot \mathrm{IF_p} = \exists \cdot \mathrm{IF_t} = \mathrm{P}$.

Theorem 13. *The following statements are equivalent:*
(1) $\mathrm{P} = \mathrm{NP}$.
(2) $\mathrm{IF_p} = \#\mathrm{P}$.
(3) $\mathrm{IF_t} = \#\mathrm{P}$.
(4) Every P*-decidable partial p-order has efficient adjacency checks.*
(5) Every P*-decidable total p-order has efficient adjacency checks.*

We know from Theorem 6 that $\mathrm{FP} \subseteq \mathrm{IF_t}$. However, the equality of these classes, or even $\mathrm{IF_t} \subseteq \mathrm{UPSV_t}$, has severe consequences.

Theorem 14. *1.* $\mathrm{FP} = \mathrm{IF_t}$ *if and only if* $\mathrm{P} = \mathrm{PP}$.
2. $\mathrm{IF_t} \subseteq \mathrm{UPSV_t}$ *if and only if* $\mathrm{UP} = \mathrm{PP}$.
3. $\mathrm{UPSV_t} \subseteq \mathrm{IF_p}$ *if and only if* $\mathrm{P} = \mathrm{UP} \cap \mathrm{coUP}$.

However the class $\mathrm{UPSV_t}$ is included in $\mathrm{IF_t}$ when restricted to strictly positive functions.

Theorem 15. $\mathrm{UPSV_t} \cap \mathrm{Nonzero} \subseteq \mathrm{IF_t} \cap \mathrm{Nonzero}$.

Corollary 16. $\mathrm{UPSV_t} \subseteq \mathrm{IF_t} - 1$.

By Corollary 9 the class $\mathrm{IF_p}$ is closed under increment. This is also true for the class $\mathrm{IF_t}$.

Theorem 17. *The class* $\mathrm{IF_t}$ *is closed under increment.*

Corollary 18. $\mathrm{IF_t} \subseteq \mathrm{IF_t} - 1$.

Though $\mathrm{UPSV_t} = \mathrm{IF_t}$ is very unlikely (see Theorem 14), for the case of strictly positive, polynomially bounded functions this actually holds. We define $\mathrm{Polybounded} =_{\mathrm{def}} \{f \mid f(x) \leq p(|x|) \text{ for some polynomial } p \text{ and all } x\}$.

Theorem 19. *1.* $\mathrm{IF_t} \cap \mathrm{Polybounded} \subseteq \mathrm{UPSV_t} \cap \mathrm{Polybounded}$.
2. $\mathrm{IF_t} \cap \mathrm{Polybounded} \cap \mathrm{Nonzero} = \mathrm{UPSV_t} \cap \mathrm{Polybounded} \cap \mathrm{Nonzero}$.
3. $\mathrm{UPSV_t} \cap \mathrm{Polybounded} \subseteq \mathrm{IF_p} \cap \mathrm{Polybounded}$ *if and only if* $\mathrm{P} = \mathrm{UP} \cap \mathrm{coUP}$.

From Theorem 5 we know that total and partial p-orders that are efficiently decidable describe the same class of functions in our setting, namely #P. If we consider total and partial p-orders that additionally have efficient adjacency checks, then this does not hold unless an unexpected complexity class collapse occurs.

Corollary 20. *If* $\mathrm{IF_t} = \mathrm{IF_p}$ *then* $\mathrm{UP} = \mathrm{PH}$.

6 Arbitrary Orders with Efficient Adjacency Checks

In the previous section we studied interval counting for polynomial-time orders with efficient adjacency checks. But what happens if we do not insist on polynomial-time decidability for the order but do keep the efficient adjacency checks? The class $\mathrm{IF}_{\mathrm{p}}^*$ ($\mathrm{IF}_{\mathrm{t}}^*$, resp.) is defined as the class of all functions $f : \Sigma^* \to \mathbb{N}$ for which there exist a partial (total, resp.) p-order A having efficient adjacency checks, and functions $b, t \in \mathrm{FP}$, such that $f(x) = \|\{z \mid b(x) <_A z <_A t(x)\}\|$ for every $x \in \Sigma^*$.

We obtain the following inclusions between the interval size classes and other complexity classes of functions. Recall that FPSPACE(poly) denotes the class of all total functions that are polynomial-space computable and polynomially length-bounded.

Proposition 21. $\mathrm{IF}_{\mathrm{t}} \subseteq \mathrm{IF}_{\mathrm{t}}^* \subseteq \mathrm{IF}_{\mathrm{p}} \subseteq \mathrm{IF}_{\mathrm{p}}^* \subseteq \#\mathrm{P} \subseteq \mathrm{FPSPACE(poly)}$.

The following theorem shows that the computational power of the class $\mathrm{IF}_{\mathrm{p}}^*$ is close to the power of FPSPACE(poly).

Theorem 22. $\mathrm{FPSPACE(poly)} - \mathrm{FP} = \mathrm{IF}_{\mathrm{p}}^* - \mathrm{FP}$.

Proof. Since $\mathrm{IF}_{\mathrm{p}}^* \subseteq \mathrm{FPSPACE(poly)}$ we have to prove only $\mathrm{FPSPACE(poly)} \subseteq \mathrm{IF}_{\mathrm{p}}^* - \mathrm{FP}$.

Let $f : \Sigma^* \to \mathbb{N}$ be in FPSPACE(poly). Without loss of generality we can assume that f is computed by a deterministic polynomial-work-space Turing machine M that makes its output in unary. Let p a strictly increasing polynomial such that every configuration K of M on input x can be encoded in an obvious way as $\langle x, K\rangle \in \Sigma^{p(|x|)}$, and that the running time of M on input x is less than $2^{p(|x|)}$. Let K_x be the initial configuration of M on input x, and let K_* be the (without loss of generality) only halting configuration of M.

Now we construct a partial p-order A on Σ^* as follows: For $i < 2^{p(|x|)}$ let $\mathrm{bin}(x, i)$ be the binary description of i with $p(|x|) + 1$ bits. The only adjacency relations of A are:

- $x\langle x, K\rangle \mathrm{bin}(x, t)0 \prec_A x\langle x, K'\rangle \mathrm{bin}(x, t+1)0$ for all $x \in \Sigma^*$, all $t < 2^{p(|x|)} - 1$, all configurations K without output, and K' being the successor configuration of K,
- $x\langle x, K\rangle \mathrm{bin}(x, t)0 \prec_A x\langle x, K\rangle \mathrm{bin}(x, t)1 \prec_A x\langle x, K'\rangle \mathrm{bin}(x, t+1)0$ for all $x \in \Sigma^*$, all $t < 2^{p(|x|)} - 1$, all configurations K with output, and K' being the successor configuration of K, and
- $x\langle x, K_*\rangle \mathrm{bin}(x, t)0 \prec_A x\langle x, K_*\rangle \mathrm{bin}(x, t+1)0$ for all $x \in \Sigma^*$ and all $t \leq 2^{p(|x|)}$.

Obviously, A has efficient adjacency checks and we have $\|\{w \mid x\langle x, K_x\rangle \mathrm{bin}(x, 0)0 <_A w <_A \langle x, K_*\rangle \mathrm{bin}(x, 2^{p(|x|)} - 1)0\}\| = f(x) + 2^{p(|x|)} - 2$. □

Using analogous proof ideas we can show still more.

Theorem 23. $\exists \cdot \mathrm{IF}_{\mathrm{p}}^* = \mathrm{PSPACE}$.

As a consequence of Theorem 22 and Theorem 23 we obtain a collection of characterizations of the class FPSPACE(poly) in terms of $\mathrm{IF}_{\mathrm{p}}^{*}$. For classes $\mathcal{F}$ and $\mathcal{G}$ of functions mapping from Σ^* to $\mathbb{N}$, let $\mathcal{F} \ominus \mathcal{G}$ denote the class of all total, nonnegative functions in $\mathcal{F}$ - $\mathcal{G}$, i.e., all total functions h for which there exist functions $f \in \mathcal{F}$ and $g \in \mathcal{G}$ such that $f(x) \geq g(x)$ and $h(x) = f(x) - g(x)$ for every x.

Corollary 24. $\mathrm{FPSPACE(poly)} = \mathrm{IF}_{\mathrm{p}}^{*} \ominus \mathrm{FP} = \mathrm{FP}^{\mathrm{IF}_{\mathrm{p}}^{*}} = \mathrm{FP}^{\exists \cdot \mathrm{IF}_{\mathrm{p}}^{*}}$.

Theorem 22 shows that $\mathrm{IF}_{\mathrm{p}}^{*}$ is as powerful as FPSPACE(poly) within the flexibility of subtracting FP functions. The next theorem shows that it is unlikely that FPSPACE(poly) is included in $\mathrm{IF}_{\mathrm{p}}^{*}$ itself.

Theorem 25. *If* $\mathrm{FPSPACE(poly)} \subseteq \mathrm{IF}_{\mathrm{p}}^{*}$ *then* $\mathrm{UP} = \mathrm{PSPACE}$.

From Theorem 22 and Theorem 23 we also get strong consequences for the case that $\mathrm{IF}_{\mathrm{p}}^{*} = \mathrm{IF}_{\mathrm{p}}$ or $\mathrm{IF}_{\mathrm{p}}^{*} \subseteq \#\mathrm{P}$ - FP.

Corollary 26. *The following statements are equivalent:*

(1) $\mathrm{P} = \mathrm{PSPACE}$.
(2) $\mathrm{IF}_{\mathrm{p}} = \mathrm{IF}_{\mathrm{p}}^{*}$.
(3) Every partial p-order with efficient adjacency checks is P*-decidable.*

Corollary 27. *1. If* $\mathrm{IF}_{\mathrm{p}}^{*} \subseteq \#\mathrm{P}$ - FP *then* $\mathrm{SPP} = \mathrm{PSPACE}$.
2. If $\mathrm{IF}_{\mathrm{p}}^{*} \subseteq \#\mathrm{P}$ *then* $\mathrm{NP} = \mathrm{SPP} = \mathrm{PSPACE}$.

Similarly to the case of IF_{t} and IF_{p}, for function classes with p-orders that only need to have efficient adjacency checks, it makes a difference whether we consider total or partial p-orders.

Theorem 28. *If* $\mathrm{IF}_{\mathrm{t}}^{*} = \mathrm{IF}_{\mathrm{p}}^{*}$ *then* $\mathrm{UP} = \mathrm{PH}$.

Figure 1 summarizes the results we have obtained regarding the inclusion structure of the considered classes. Though we have not proven consequences for the case of further collapses of classes that are drawn in the figure as being different, we conjecture that the inclusions in the figure are all one can prove without assuming unexpected collapses of complexity classes.

7 The Complexity of Counting Divisors

Finally, we explore the complexity of the function $\#\mathrm{DIV} : \mathbb{N} \to \mathbb{N}$ defined as

$$\#\mathrm{DIV}(m) =_{\mathrm{def}} \|\{n \in \mathbb{N} \mid n \neq 1, n \neq m, \text{ and } n \text{ divides } m\}\|$$

for $m \geq 1$ and $\#\mathrm{DIV}(0) =_{\mathrm{def}} 0$, and we relate it to the interval size class IF_{p}.

The complexity of #DIV depends on the complexity of primality testing. Let PRIMES be the set of all prime numbers. It is well known that PRIMES is in the class BPP but it is not known that PRIMES is in P (though sometimes conjectured, see, e.g., [17,2]). The next result shows that this is necessary and sufficient for #DIV to belong to the interval size class IF_{p}.

Theorem 29. PRIMES *is in* P *if and only if* #DIV *is in* IF_{p}.

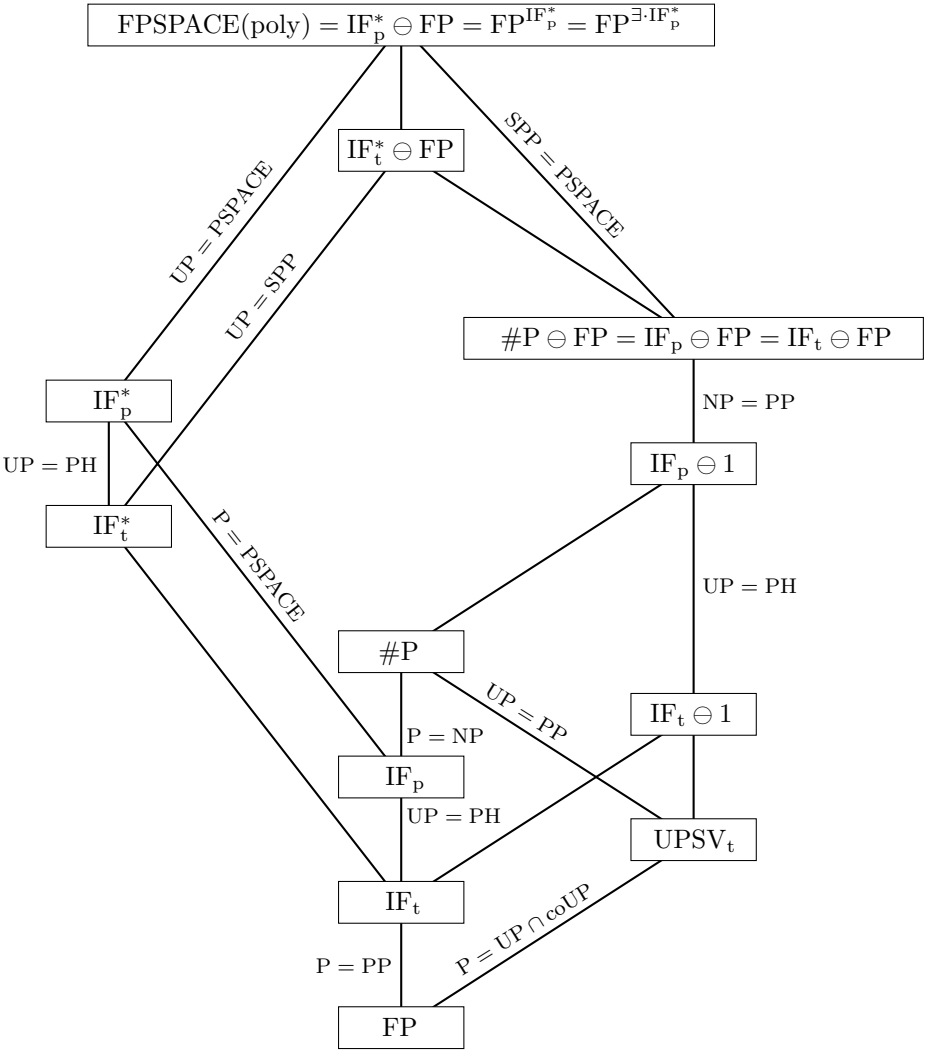


Fig. 1. The landscape of interval size classes and further function classes. An equation E at the edge between the function classes $\mathcal{F}_1$ and $\mathcal{F}_2$ means: If $\mathcal{F}_1 = \mathcal{F}_2$ then E. Edge-equations that are not immediate consequences of the results of this paper are well-known or easy to see. Since FP is of type $\Sigma^* \to \mathbb{N}$ and the edges indicate containment claims, the fact that above we use "$\ominus$" rather than "$-$" is not of any consequence.

Proof. [⇒]: Define A to be the set of all (x, y) with x dividing y. Then clearly $A \in \mathrm{P}$ and A is a partial p-order (see the discussion in the introductory section). Moreover, it clearly holds that $x \neq 0$ and y are adjacent with respect to A if and only if $\frac{y}{x}$ is a prime number. So, since PRIMES $\in$ P, adjacency checking is easy here. Consequently, #DIV is in $\mathrm{IF_p}$.

[⇐]: Let #DIV $\in \mathrm{IF_p}$ via partial p-order $A \in \mathrm{P}$ having efficient adjacency checks and functions $b, t \in \mathrm{FP}$. Then an $x > 1$ is a prime number if and only if $\#\mathrm{DIV}(x) = 0$, and this is the case if and only if $b(x) \not\preceq_A t(x)$, or $b(x) \prec_A t(x)$, or $b(x) = t(x)$. The latter conditions can be checked in polynomial time depending on $|x|$. Hence, PRIMES belongs to P. □

8 Conclusions

Reviewing all the results on the interval size classes $\mathrm{IF_p}$, $\mathrm{IF_p^*}$, $\mathrm{IF_t}$, and $\mathrm{IF_t^*}$, it seems that we have a good idea of the computational power of the classes $\mathrm{IF_p}$ and $\mathrm{IF_p^*}$, but there is a lack of corresponding knowledge for $\mathrm{IF_t}$ and $\mathrm{IF_t^*}$. Can we find out more? In particular, what about the classes $\exists\cdot\mathrm{IF_t^*}$ and $\exists\cdot(\mathrm{IF_t^*} - \mathrm{FP})$, which can be considered to be a kind of "total order" PSPACE, and what about the class $\mathrm{IF_t} - \mathcal{O}(1)$, which can be considered to be a kind of "total order" #P?

References

1. D. P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* International Series in Computer Science. Prentice Hall, New York, 1994.
2. A. E. F. Clementi, J. D. P. Rolim, and L. Trevisan. Recent advances towards proving P=BPP. *Bulletin of the EATCS*, 64:96–103, 1998.
3. S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings 3rd ACM Symposium on Theory of Computing*, pages 151–158, 1971.
4. S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48:116–148, 1994.
5. J. Gill. Computational complexity of probabilistic complexity classes. *SIAM Journal on Computing*, 6:675–695, 1977.
6. A. V. Goldberg and M. Sipser. Compression and ranking. *SIAM Journal on Computing*, 20(3):524–536, 1991.
7. J. Goldsmith, L. A. Hemachandra, D. Joseph, and P. Young. Near-testable sets. *SIAM Journal on Computing*, 20(3):506–523, 1991.
8. J. Grollmann and A. L. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335, 1988.
9. H. Hempel and G. Wechsung. The operators min and max on the polynomial hierarchy. *International Journal of Foundations of Computer Science*, 11(2):315–342, 2000.
10. U. Hertrampf, H. Vollmer, and K. W. Wagner. On balanced versus unbalanced computation trees. *Mathematical Systems Theory*, 29:411–421, 1996.
11. K. Ko. On self-reducibility and weak P-selectivity. *Journal of Computer and System Sciences*, 26:209–221, 1983.
12. S. Kosub. A note on unambiguous function classes. *Information Processing Letters*, 72(5-6):197–203, 1999.

13. R. E. Ladner. Polynomial space counting problems. *SIAM Journal on Computing*, 18(6):1087–1097, 1989.
14. L. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973.
15. A. R. Meyer and M. Paterson. With what frequency are apparently intractable problems difficult? Technical Report MIT/LCS/TM-126, Laboratory for Computer Science, MIT, Cambridge, MA, 1979.
16. A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proceedings 13th Symposium on Switching and Automata Theory*, pages 125–129. IEEE Computer Society Press, Los Alamitos, 1972.
17. G. L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13:300–317, 1976.
18. M. Ogiwara and L. A. Hemachandra. A complexity theory of feasible closure properties. *Journal of Computer and System Sciences*, 46:295–325, 1993.
19. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, 1994.
20. J. Simon. *On Some Central Problems in Computational Complexity*. PhD thesis, Cornell University, Ithaca, 1975.
21. L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.
22. L. G. Valiant. Relative complexity of checking and evaluation. *Information Processing Letters*, 5:20–23, 1976.
23. L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):411–421, 1979.
24. H. Vollmer and K. W. Wagner. Complexity classes of optimization functions. *Information and Computation*, 120:198–219, 1995.

Communication Gap for Finite Memory Devices*

Tomasz Jurdziński[1,2] and Mirosław Kutyłowski[3,4]

[1] Institute of Computer Science, Wrocław University
[2] Department of Computer Science, Technical University of Chemnitz
[3] Department of Mathematics and Computer Science, Poznań University
[4] Institute of Mathematics, Wrocław University of Technology

Abstract. So far, not much is known on communication issues for computations on distributed systems, where the components are weak and simultaneously the communication bandwidth is severely limited. We consider synchronous systems consisting of finite automata which communicate by sending messages while working on a shared read-only data. We consider the number of messages necessary to recognize a language as a its complexity measure.
We present an interesting phenomenon that the systems described require either a constant number of messages or at least $\Omega((\log\log\log n)^c)$ of them (in the worst case) for input data of length n and some constant c. Thus, in the hierarchy of message complexity classes there is a gap between the languages requiring only $O(1)$ messages and those that need a non-constant number of messages. We show a similar result for systems of one-way automata. In this case, there is no language that requires $\omega(1)$ and $o(\log n)$ messages (in the worst case). These results hold for any fixed number of automata in the system.
The lower bound arguments presented in this paper depend very much on results concerning solving systems of diophantine equations and inequalities.

1 Introduction

Traditional complexity theory usually disregards either communication between processing units or their internal limitations. On the other hands, for many reasons it is interesting to consider the situation where we have many weak processing units and severely limited communication between them. Not much is known about communication complexity of problems in this setting. The classical complexity theory concentrates on the situation when input data is divided into parts and every part is available to exactly one participant, while local computational resources are unlimited. Perhaps, the major achievement for restricted local resources are the results of Beame et al. [1] on tradeoffs between the communication volume and space used.

* This research was partially supported by Deutsche Forschungsgemeinschaft and Komitet Badań Naukowych, grants 8 T11C 032 15, 8 T11C 012 18, and finally 7 T11C 032 20. Some of the results are contained in the Ph.D. dissertation of the first author. Some work has been done when the second author was visiting University of Mannheim.

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 1052–1064, 2001.

We consider a setting, in which we do not use partitioning of input data for getting lower bounds. While the whole input is available to every processing unit we show that internal limitations of processing units cause large communication requirements.

Model. We consider the simplest model one can imagine to study the problem mentioned above, namely systems of (a finite number of) finite automata (equivalent to "multihead finite automata" [5], see also [12]). The automata work independently on a common read-only input data and communicate by sending (broadcasting) messages. Since each automaton has only finitely many internal states, the number of different messages, as well as their size, is constant with respect to the size of the data. So the number of messages corresponds roughly to communication volume. The transition function of a single automaton depends on its internal state, an input symbol seen and the messages received from other automata at this moment. The automata work in parallel and synchronously, the messages are delivered instantly. More formally:

Definition 1. *We say that $M = (A_1, A_2, \ldots, A_k)$ is a system of finite automata, if there is a finite set Q such that every internal state of any A_i is in Q; each automaton has a single read-only head; and a step of M is executed as follows: if A_i, for $i \in \{1, \ldots, k\}$, is in a state q and a is an input symbol seen by the read-head of A_i, then:*
(1) A_i broadcasts a message $m = \mu_i(q, a)$ to other automata (for $\mu_i(q, a) = nil$, no message is sent by A_i);
(2) A_i changes its internal state to q' and moves its read head r positions to the right ($r \in \{0, 1, -1\}$), where q' and r are given by transition function δ_i:

$$(q', r) = \delta_i(q, a, m_1, \ldots, m_{i-1}, m_{i+1}, \ldots, m_k),$$

m_j denotes the message sent by automaton A_j during the first phase of the step or "nil", if no such a message is sent.

Communication configuration *is a configuration of the system in which at least one automaton sends a message (and the final configuration – the configuration in which every automaton is in the* ***final state****, i.e., in the state for which no transition is defined, independently of messages received and symbol observed). An* ***initial configuration*** *of the system M is the configuration in which all automata see the leftmost cell of the input tape and are in initial states $q_0^1, \ldots, q_0^k$. An input word x is given on the tape together with endmarkers $\triangleright$ and $\triangleleft$ preventing the heads to leave $\triangleright\, x \,\triangleleft$. A system accepts an input x when a fixed automaton (for instance the first one) enters an accepting state (which is the final state) during the computation on $\triangleright\, x \,\triangleleft$ which starts in the initial configuration. A system recognizes language L, when a fixed automaton accepts input x if and only if $x \in L$.*

Message complexity classes. Let MESSAGE($f(n)$) denote the class of languages that may be recognized by a system of automata such that at most $f(n)$ communication configurations occur for computation of the system on every input word of length n (observe that the number of messages sent is at

most k times greater than the number of communication configurations). Note that already MESSAGE(1) contains a non-regular language $\{a^m b^m | m \in \mathbb{N}\}$ and a non-context-free language $\{a^m b^m c^m | m \in \mathbb{N}\}$. For example, to recognize $\{a^m b^m | m \in \mathbb{N}\}$ by 2 automata and one message we start both automata moving right in such a way that the automaton A_2 makes two steps on each symbol and the automaton A_1 makes one step on each symbol (so A_2 moves two times faster than A_1). Additionally, A_2 checks if the input has the form a^*b^*. Once A_2 arrives at the right endmarker, it sends the message. The input is accepted if and only if A_1 receives the message just in the moment when it is on the border between a's and b's.

A number of hierarchy results concerning message complexity classes have been obtained in [8]. While there is a strict hierarchy concerning a constant number of communication configurations (or messages), for non-constant number of messages, the hierarchy of this kind starts from MESSAGE($\log\log n$). It is also known [8] that the language
$L_2 = \{1^{f_1}\#1^{f_2}\#\ldots\#1^{f_k} : f_1 = 2, f_2 = 3, f_{i-1}|(f_i - 1), f_{i-2}|(f_i - 1), f_i > 1 \text{ for } i = 3, \ldots, k\}$
is in MESSAGE($O(\log\log n)$) but not in any class of the type MESSAGE($o(\log\log n)$).

2 New Results

We show a gap phenomenon for message complexity classes: if the number of messages is non-constant, it must reach a certain level. Results of this kind are not sparse in complexity theory, but they are a little bit surprising in the context of communication complexity (note that for the classical communication complexity it does not occur). Our proof method relies on reducing the problems to questions about the size of minimal solutions of some Diophantine systems. In references in [11] one can find more examples of reductions of complexity theory problems to "Diophantine problems".

For one-way automata (for which the heads cannot move to the left) we show that there is a gap between $\Theta(1)$ and $\Theta(\log n)$ message complexity classes:

Theorem 1. *For a function $f(n)$ such that $f(n) = \omega(1)$ and $f(n) = o(\log n)$, there is no* one-way *multi-automata system which requires $\Theta(f(n))$ communication configurations.*

The bound $\log n$ is optimal, since it is easy to see that the language $L_1 = \{1^{2^0}\#1^{2^1}\#1^{2^2}\#\ldots\#1^{2^k} : k \in \mathbb{N}\}$ can be recognized with $O(\log n)$ messages: (the algorithm is based on the trick with different "speeds" presented above for the language $\{a^m b^m | m \in \mathbb{N}\}$). For two-way systems, we have a much smaller bound (note that it may not exceed $\log\log n$, as shown by the language L_2).

Theorem 2. *There is a constant c such that for $f(n) = \omega(1)$ and $f(n) = o((\log\log\log n)^c)$, there is no* two-way *multi-automata system which requires $\Theta(f(n))$ communication configurations.*

Outline of the proofs. Let S be a system of finite automata that needs $\omega(1)$ communication configurations. Our goal is to show that if there is an input that requires exactly g communication configurations, then there is an input of length $O(2^g)$ (in the case of one-way systems) or $2^{2^{2^{u(g)}}}$ for some polynomial u (in the case of two-way systems) with a similar history of computation, and, what is most important, for which g communication configurations occur. Then, Theorems 1 and 2 follow immediately.

The first step in finding such inputs is to show that after dividing computations into some number of categories, each category can be described by a diophantine system. Such a system consists of linear equations and inequalities with integer coefficients with absolute values bounded by a constant (in the the case of one-way automata) and, additionally, of expressions like $f(x)|g(x)$, where f and g are linear functions with integer coefficients bounded by a constant (in the general case).

The most important feature of diophantine systems derived is that any computation corresponds to some diophantine solution of the system (i.e., a solution in $\mathbb{N}$), and vice versa, every diophantine solution describes some input data. Additionally, the input length is bounded by a linear expression in some of the variables occurring in the diophantine system.

To complete the proof, we argue that there is a small solution of the diophantine system. Such a solution corresponds to an input word, for which, in turn, the number g of communication configurations is relatively big, yielding the claimed bound on the number of communication configurations in the worst case. For linear equations and inequalities with coefficients having absolute values bounded by a constant the minimal solution is at most exponential in the size of the system [4] (see [3] for efficient methods for finding minimal solutions). For the case of divisibilities, one may analyze the construction of Lipshitz [9] to find that the minimal solution is at most triple exponential.

The key technical part of this paper (Section 3) is expressing behavior of systems of automata by diophantine systems. In Section 4, we guide over the results concerning the size of minimal solutions of diophantine systems.

3 Reduction to Diophantine Problems

First, we introduce some notions describing work of automata during periods when no messages are sent. Let $\{A_i\}_{i=1,\ldots,k}$ be a set of finite automata of the system S, let Q be the set of their states. **Transition** is a tuple (A_i, q, s, q', s') where $q, q' \in Q$ and $s, s' \in \{Left, Right\}$. Word w satisfies a transition (A_i, q, s, q', s'), if automaton A_i, starting the computation in state q on the side s of w, leaves the word w in state q' on the side s' (if it receives no message in this time) and does not send any message in this part of computation. **Clocked transition** is a tuple (A_i, q, s, q', s', t) where $q, q' \in Q$, $s, s' \in \{Left, Right\}$ and $t \in \mathbb{N}$. Word w satisfies a clocked transition (A_i, q, s, q', s', t), if A_i makes t steps in part of computation which starts when it enters w in the state q on the side s and finishes when it leaves w in the state q' on the side s' (and A_i does not leave w between these two events). **Transition set** is a set of transitions. **Clock**

characterization is a set of clocked transitions. A **communication partition** of a word x by the system S is a sequence of words $x_1, \ldots, x_{h+1}$ and a sequence of symbols $\sigma_1, \ldots, \sigma_h$ such that $x = x_1\sigma_1 x_2\sigma_2 \ldots x_h\sigma_h x_{h+1}$ and:

1. during computation on x, at each communication configuration and at the final configuration of S the read-heads occupy positions from the set $\{|x_1\sigma_1 \ldots x_i\sigma_i| : i \in \{1, \ldots, h\}\}$
2. for each $i \in \{1, \ldots, h\}$, there is a communication configuration, in which some head scans σ_i (i.e., it is on position $|x_1\sigma_1 \ldots x_i\sigma_i|$).

The positions $u = |x_1\sigma_1 \ldots x_i\sigma_i|$ for $i \in \{1, \ldots, h\}$, are called **communication positions**. Obviously, $h \leq k \cdot (g + 1)$, where k is the number of automata and g the number of communication configurations. The words $x_1, x_2, \ldots, x_{h+1}$ are called **silent blocks** of x.

A **communication pattern** of the system S on input x, is a set of tuples of the form $\langle i, (q_1, i_1), (q_2, i_2), \ldots, (q_k, i_k)\rangle$ such that when the ith communication configuration occurs the automaton A_l is in the state q_l on position $|x_1\sigma_1 \ldots x_{i_l}\sigma_{i_l}|$ for $l \leq k$. By a **full transition set** of the block x_i of the input word x we mean a transition set of x_i which consists of all transitions on x_i made during the computation of the system S on x. For any word w satisfying the full transition set T, a **full clock characterization** of w is its clock characterization consisting of its clocked transitions of all transitions from T.

A **computation digest** of the system S on word x, denoted by $H_S(x)$ consists of:

– a sequence $\sigma_1, \ldots, \sigma_h$ of symbols on consecutive communication positions of x,

– a *communication pattern* of S on x,

– *full transition sets* of all silent blocks of x.

Observe that a computation digest fixes neither the silent blocks $x_1, \ldots, x_h$ nor their full clock characterizations.

A state q is the **blocking state** for an automaton A_i and a symbol a if A_i makes no move and no change of its state when it is in state q, reads input symbol a and no message arrives, i.e., $(q, 0) = \delta_i(q, a, nil, \ldots, nil)$,

3.1 Outline of Diophantine Characterization of One-Way Automata

First, we give an idea of a "linear" characterization of all inputs with computation digest $H_S(x) = H$. Inside a silent block x_k of x no message is sent or received. So, from the point of view of computation of the system, the only important property of x_k is its full clock characterization, C_k. If any other word x'_k has the same full clock characterization C_k, then we may replace x_k with x'_k without changing messages sent (and acceptance). If x'_k is shorter than x_k, then we obtain a shorter input with computation digest H. However, in most cases it would be impossible to shorten one block only, without changing the computation digest. Our approach is to describe dependencies between the blocks and then to shorten the blocks simultaneously keeping these dependencies valid.

Example. To see how characterizations work, we consider a system S with two automata A_1 and A_2 on the word x that has a communication partition which consists of symbols $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ and silent blocks x_1, x_2, x_3, x_4. The automaton A_1 (respectively A_2) visits communication positions in states q_1, q_2, q_3, q_4 (respectively q'_1, q'_2, q'_3, q'_4). The *communication pattern* of S on x equals $\{\langle 1, (q_2, 2), (q'_1, 1)\rangle, \langle 2, (q_3, 3), (q'_2, 2)\rangle, \langle 3, (q_4, 4), (q'_4, 4)\rangle\}$. Assume also that automaton A_1 (respectively A_2) sends messages only in states q_2 and q_4 (respectively q'_2), and q_3 is the only blocking state. Let $t_{1i}, \ldots, t_{4i}$ denote the numbers of steps spent by automaton A_i on blocks $x_1, \ldots, x_4$ for $i \in \{1, 2\}$. We may express H, the computation digest of S on x, by full transition sets of blocks x_1, x_2, x_3, x_4 and the conditions:

$$t_{11} + t_{21} = t_{12}, \quad t_{22} \geq t_{31}, \quad t_{41} = t_{32} + t_{42}.$$

In general, once we have equations and inequalities as in the example above, we look for the words $x'_1, \ldots, x'_h$ which have :

– *full transition sets* determined by the computation digest H,

– *full clock characterizations* satisfying an appropriate system of linear equations and inequalities (determined by the *communication pattern*).

The equations and inequalities mentioned above describe dependencies between the numbers of steps made by the automata when they go through silent blocks. However, the variables denoting the numbers of steps made by different automata on the same block (like t_{11} and t_{12} in the example), are dependent. The main difficulty here is that the "speed" of each automaton within a block fluctuates heavily depending on the input symbols read. "Speed pattern" might be extremely complicated. However, going through different speed phases is like walking through a graph of different "speed states". This graph has a constant size, depending on automata only. So, we may hope to describe such walks by linear expressions with variables describing how many times one goes through certain cycles in the graph.

In fact, implementing this idea is nontrivial. To the full transition set of the block x_i we assign not a single set of expressions and inequalities but a family L_i of such sets. Each set $\phi \in L_i$ consists of some linear inequalities and a linear expression l_{ij} for each automaton A_j. The expression l_{ij} computes t_{ij}, the number of steps made by the automaton A_j on block i of the communication partition.

L_i describes all words with a given full transition set in the following sense: Each such a word corresponds to an assignment of variables for some $\phi \in L$ that fulfills the conditions in ϕ. Vice versa: for each set ψ of L each assignment of variables that satisfies conditions of ψ corresponds to a computation on some word. It is important for us that the length of the silent block i is upper bounded by each expression l_{ij}.

3.2 Linear Characterization Lemma for One-Way Automata

Lemma 1 (Linear Characterizations Lemma). *Let S be a system of k one-way automata, let H be a computation digest of S on some input x and g communication configurations occur during a computation described by H. We can*

associate with H a family L of systems of linear diophantine equations and inequalities such that for any system of equations $Ay = b$ and inequalities $Cy \geq d$ in L the following conditions hold:

1. *the number of equations and inequalities occurring in the system is $O(g)$,*
2. *the number of variables of the system is $O(g)$,*
3. *coefficients of A, b, C and d are integers with absolute values limited by some constant c_2 which depends only on k and the number of the states of automata,*
4. *let $x' = x'_1\sigma_1 \ldots x'_h\sigma_h$ be any word such that its computation digest is H, communication partition consists of symbols $\sigma_1, \ldots, \sigma_h$ and blocks $x'_1, \ldots, x'_h$. Let*

$$\{\langle A_i, q_{ij}, s_{ij}, q'_{ij}, s'_{ij}, t_{ij}(x')\rangle \mid 1 \leq i \leq k\}$$

denote the full clock characterization of word x'_j [1] (so $t_{ij}(x')$ denotes the number of steps made by A_i on x'_j, the j-th silent block). Then, there are $\gamma_1, \ldots, \gamma_m \in \mathbb{N}$, which form a solution of some system in L such that $t_{ij}(x') = \gamma_{(j-1)k+i}$ for $1 \leq i \leq k$, $1 \leq j \leq h$. Conversely, for each solution $\gamma_1, \ldots, \gamma_m$ of any system in L one can find a word w with computation digest H such that $t_{ij}(w) = \gamma_{(j-1)k+1}$ for $1 \leq i \leq k$, $1 \leq j \leq h$ (note that m may be bigger than kh).

The rest of this section is devoted to the proof of Linear Characterizations Lemma. The key part is to describe dependencies between number of steps necessary to read the same silent block by different automata. We achieve this in the following lemma:

Lemma 2. *Let $T = \{\langle A_i, q_i, s_i, q'_i, s'_i\rangle | i = 1, \ldots, k\}$ be a transition set, $X = X(T)$ be a set of words satisfying transitions from the set T. Let $\{\langle A_i, q_i, s_i, q'_i, s'_i, \alpha_i(x)\rangle | i = 1, \ldots, k\}$ denote the full clock characterization of a word x, for $x \in X$. Then, there exists a family R of sequences of linear expressions and inequalities of c variables (c and the number of elements of R are bounded by a number which depends on k and the number of states of the automata only) such that:*

1. *Every sequence $z \in R$ consists of k expressions $z_1, \ldots, z_k$ and c inequalities $n_1, \ldots, n_c$ of variables $v_1, \ldots, v_c$. Moreover, each inequality has the form $v_i \geq c_i$, for $c_i \in \{0, 1\}$.*
2. *For every $x \in X$, there is a sequence of linear expressions and inequalities $z \in R$,*
   $z = \langle z_1, \ldots, z_k; n_1, \ldots, n_c\rangle$ and numbers $d_1, \ldots, d_c \in \mathbb{N}$ such that $\alpha_i(x) = z_i(d_1, \ldots, d_c)$ for $i = 1, \ldots, k$ and $n_i(d_1, \ldots, d_c)$ is satisfied, for $1 \leq i \leq c$.
3. *For every $d_1, \ldots, d_c \in \mathbb{N}$ and each $z = \langle z_1, \ldots, z_k; n_1, \ldots, n_k\rangle \in R$, if $n_i(d_1, \ldots, d_c)$ holds for $1 \leq i \leq c$, then there is a word $x \in X$ such that $\alpha_i(x) = z_i(d_1, \ldots, d_c)$, for $1 \leq i \leq k$.*

Proof. We define an auxiliary one-way finite automaton A with the set of states Q^k and initial state $(q_0^1, \ldots, q_0^k)$. It works as follows. If A is in a state $\langle q_1, \ldots, q_k\rangle$

[1] there is only one tuple for every automaton, since we consider one-way devices

on the symbol a which is not an endmarker, then in one step it moves right and reaches a state $\langle q'_1, \ldots, q'_k \rangle$ such that for $i \in \{1, \ldots, k\}$, if the automaton A_i starts the computation in the state q_i on the symbol a, then it enters the next cell in state q'_i. Every transition of A given by $\delta(\langle q_1, \ldots, q_k \rangle, a) = (\langle q'_1, \ldots, q'_k \rangle, 1)$ is associated with a *step vector* $(s_1, \ldots, s_k)$ such that for $i \leq k$ automaton A_i needs s_i steps in order to move right from the state q_i while reading the symbol a. Now, we define a directed multi-graph G representing the transition function of A and its steps vectors. The vertices of the graph represent the states of A; the edges represent transitions of A. Each edge is labeled by an input symbol a and a step vector $(s_1, \ldots, s_k)$ associated to this transition. We identify an edge by a tuple (v_1, v_2, a), where v_1, v_2 are vertices adjacent to the edge (v_1 is the source) and a is a symbol labeling the edge. For an edge (v_1, v_2, a), for $i \in \{1, \ldots, k\}$, let $s_i(v_1, v_2, a)$ denote the ith coefficient of the *steps vector* of this edge.

Computation of A on input $x = x(1)x(2)\ldots x(p)$ (hence, also of $A_1, \ldots, A_k$) is represented by a path $W = (w_1, w_2, x(1))(w_2, w_3, x(2)) \ \ldots \ (w_p, w_{p+1}, x(p))$ in G which starts and finishes in vertices representing appropriate states, say vertices s and t. Then, A_i makes $\Sigma_{j=1}^{p} s_i(w_j, w_{j+1}, x(j))$ steps on x. We denote this number by $s_i(W)$ or $s_i(x)$.

Our goal is to construct a family R of sequences of expressions and inequalities over some fixed set of variables V which "linearly characterize" all paths from s to t. Moreover, the absolute values of coefficients in expressions and inequalities as well as the number of elements of R are to be bounded by numbers which depend only on the size of G. It means that for every path W from s to t there is a sequence $\langle z_1, \ldots, z_k, n_1, \ldots, n_c \rangle \in R$ and values $\mathcal{V}$ for variables V such that inequalities $n_1, \ldots, n_c$ are satisfied for $\mathcal{V}$ and $s_i(W) = z_i(\mathcal{V})$ for each i.

The proof is by induction on the number of vertices in G. We partition each path into subpaths going into the graph $G' = G \backslash \{s\}$. Then, using induction hypothesis we combine "linear descriptions" of paths in G' into a description of paths from s to t in G.

The claim is obviously true for graphs with one vertex. Now, assume that it holds for all graphs with less than n vertices. Let G be a graph with n vertices, $U = (u_1, u_2, \delta_1) \ \ldots (u_{p-1}, u_p, \delta_{p-1})$ be a path in G, $u_1 = s$ and $u_p = t$. Let $U = U_1 U_2 \ldots U_a$ be the coarsest partition of U such that s may occur only on the first and the last edge of any U_i. So for every i, the path U_i without the last and the first edge does not leave the graph $G' = G \backslash \{s\}$. In this way we partition any path U from s to t into subpaths going through the graph G' with a smaller number of vertices and some edges "connecting" these subpaths. By the induction hypothesis, these subpaths have appropriate "linear characterizations". Our strategy is to use them for constructing such a "linear characterization" for paths from s to t in G. Obviously, the linear characterization should have the size bounded by some value dependent on the size of the graph, independent of the lengths of paths. However, the partition of U may depend on its length. In order to overcome this problem, we group subpaths in categories. A category is defined by:

(a) the first and the last edge: (s, u_1, σ_1) and (u_2, v, σ_2) (where $v \in \{s, t\}$) of the subpaths of this category,

(b) a sequence of linear expressions and inequalities $r = \langle w_1, \ldots, w_k; n_1, \ldots, n_m \rangle$

for paths from u_1 to u_2 in G' (such a sequence exists by the induction hypothesis).
Observe that, by the induction hypothesis, the number of categories is limited by a number which depends solely on the size of the graph (so, on the number of states of automata). Let $cat(U_i)$ denote the category of the subpath U_i, $K(U)$ denote the set of all categories of subpaths $U_1, \ldots, U_a$.

Now, we show how to construct a sequence $\langle w_1, \ldots, w_k; n_1, \ldots, n_m \rangle$ of equalities and inequalities which characterizes all paths U' from s to t in G such that $K(U') = K(U)$. So, by considering all possible sets of categories we obtain a family of sequences which linearly characterize all possible paths from s to t in G.

Let $K(U) = \{K_1, \ldots, K_p\}$. We take one category, say K_l, and construct the sequence of expressions and inequalities which will "cover" all subpaths of this category in U and similarly all subpaths of this category in every path U' from s to t such that $K(U') = K(U)$. Assume that the category K_l is characterized by:
(a) edges (s, u_1, σ_1) and (u_2, s, σ_2) (we omit the case in which the subpath terminates in t; it can be considered in a similar way),
(b) linear expressions $w_i = a_{i0} + \sum_{j=1}^{m} a_{ij}\xi_j$ for $i \in \{1, \ldots, k\}$, and inequalities $n_i \equiv (\xi_i \geq c_i)$, for $i \in \{1, \ldots, d\}$ where $c_i \in \{0, 1\}$, over variables $\xi_1, \ldots, \xi_m$.

Let $B = \{i \mid c_i = 1\}$. Our goal now is to construct expressions and inequalities such that all steps of the automata made "inside" subpaths of category K_l will be "counted" by these expressions. The key point is, we do not care where these subpaths occur, and how they exactly look like. We only count the total number of steps inside them. We claim that inequalities $\quad n'_0 \equiv (\xi_0 \geq 1), \quad n'_i \equiv (\xi_i \geq 0)$ for $i \in \{1, \ldots, m\}$, and expressions

$$w'_i = \left(a_{i0} + \sum_{j \in B} a_{ij} + s_i(s, u_1, \sigma_1) + s_i(u_2, s, \sigma_2) \right) \xi_0 + \sum_{j=1}^{m} a_{ij}\xi_j$$

over variables $\xi_0, \ldots, \xi_m$ may be used for this purpose. Intuitively, the "new" variable ξ_0 denotes the number of subpaths of category K_l in the path; elements summarized as a coefficient multiplied by ξ_0 consist of all "elements" which have to occur in every (sub)path of category K_l. The remaining variables are responsible for the "optional" parts of the paths (possibly repeated many times).

Finally, we join expressions and inequalities of all categories $K_1, \ldots, K_p$.

Let N_i denote the sequence of inequalities and $W_i = \langle w_i^1, \ldots, w_i^k \rangle$ be the sequence of expressions which characterize the subpaths of category K_i (to avoid conflicts, each category has its own variables). The sequence of inequalities and expressions which characterizes the set of paths U' from s to t such that $K(U') = K(U)$ contains the inequalities from N_i, over all i, and the expressions $\sum_i w_i^1, \ldots, \sum_i w_i^k$. It follows immediately from the construction that this sequence has all properties claimed. ∎ Lemma 2

Now, in order to complete the proof of Linear Characterization Lemma we show how to construct a system of linear equations and inequalities which defines the number of steps spent on silent blocks and dependencies between them.

Let $\langle i,(q_1,j_1),(q_2,j_2),\ldots,(q_l,j_l)\rangle$ and $\langle i+1,(q'_1,j'_1),(q'_2,j'_2),\ldots,(q'_l,j'_l)\rangle$ be consecutive entries in the communication pattern H for any $i \in \{1,\ldots,g-1\}$. Let A_α be the automaton which sends the message in the $(i+1)$st communication configuration (if more than one automaton sends a message we pick any of them). Let $I \subseteq \{1,\ldots,l\}$ be the indices of automata which are then in blocking states. Then, the properties

$$t_{\alpha,j_\alpha+1}(x)+t_{\alpha,j_\alpha+2}(x)+\ldots+t_{\alpha,j'_\alpha}(x) \geq t_{\delta,j_\delta+1}(x)+t_{\delta,j_\delta+2}(x)+\ldots+t_{\delta,j'_\delta}(x) \text{ for } \delta \in I,$$

$$t_{\alpha,j_\alpha+1}(x)+t_{\alpha,j_\alpha+2}(x)+\ldots+t_{\alpha,j'_\alpha}(x) = t_{\delta,j_\delta+1}(x)+t_{\delta,j_\delta+2}(x)+\ldots+t_{\delta,j'_\delta}(x) \text{ for } \delta \notin I$$

determine bahaviour of system S between the ith and the $(i+1)$st communication configuration. By Lemma 2, the variables t_{ij} in the above expressions may be replaced by some linear expressions of a constant number of variables (attached with some inequalities concerning these variables). This yields the inequalities stated by Lemma 2. It is easy to see that they satisfy all properties claimed.

3.3 Outline of Diophantine Characterization of Two-Way Automata

In this section, we confine ourselves to these elements that differ from the construction for the one-way case. Before we go into details, we sketch how to recognize language L_2, which witnesses that $O(\log\log n)$ messages may suffice in the case of two-way automata. Simultaneously, it explains in which direction the changes in the construction of diophantine systems must go. We use automata A_1 and A_2 to recognize L_2. First, A_1 checks if the first two blocks of ones have lengths 2 and 3. Then, for $i = 3,4,\ldots$ they verify that all required divisibilities hold. In order to check whether $f_{i-1}|(f_i-1)$ (and similarly if $f_{i-2}|(f_i-1)$), automata A_1 and A_2 start moving with the same speed through block $i-1$ and i. Automaton A_1 goes from the left to the right in block i, automaton A_2 loops between the ends of block $(i-1)$. They terminate this stage of computation when A_1 reaches symbol # and sends a message. Since A_2 may remember in states, if it has left the symbol # in the previous step, the divisibility can be easily checked.

Now, we move to the general diophantine construction. Consider the number of steps needed to move through a silent block of input x. The first problem is that each automaton may move through a silent block many times entering it from left/right and leaving on the left/right side. However, the number of possible transitions over a silent block is limited by the number of possible "starting points" determined by the state of the automaton and the side on which it enters the block. (A head cannot loop inside a silent block, since finally a message would be sent when the head is inside the block.) So, we have variables $t_{i,j,q,d}$ denoting the number of steps which the automaton A_i spends before leaving the jth silent block, when entering it in the state q from the side d.

In order to describe dependencies between the variables $t_{i,j,q,d}$, we have to use divisibilities. This is due to the fact that, as in the example above, while the automaton which sends the ith message moves through some blocks, the other automata may loop over some other blocks. For example, assume that the

ith message is sent by automaton A_α, which during the part of computation between the $(i-1)$st and the ith message makes T_α steps (T_α is obviously a sum of some values $t_{\alpha,*,*,*}$). At this time, automaton A_β, after some initial computation consisting of $T_{\beta,1}$ steps gets into a loop consisting of $T_{\beta,2} > 1$ steps (by loop we mean a part of computation which starts and finishes in state q on position p for any q and p). Obviously, one may define beginning of the loop to get that A_β is at the end of the loop when a message is sent by A_α. $T_{\beta,1}$ and $T_{\beta,2}$ are obviously sums of some values $t_{\beta,*,*,*}$. The dependencies described take the form $T_{\beta,2}|(T_\alpha - T_{\beta,1})$. In the case that A_β does not get into a loop, we express the dependencies like for systems of one-way automata.

The next major change with respect to the one-way case is how to construct linear formulas for the variables $t_{i,j,q,d}$. Let us consider a transition $\langle A_i, q, d, q', d'\rangle$ of block j. First, we construct a (nondeterministic) one-way automaton for such a transition. This automaton goes from left to right and guesses crossing sequences for consecutive positions (similarly as it is done to show equivalence between one-way and two-way finite automata [6]; crossing sequences required for border positions of the block are determined by the tuple $\langle A_i, q, d, q', d'\rangle$). With each step of this automaton we associate a weight denoting the number of steps needed by A_i to perform transitions described by the crossing sequence. The automaton is nondeterministic, but its work corresponds to a path in a certain graph. The rest of the construction is the same as before (so, we built a "Cartesian product" of such automata associated to transitions).

4 Solutions for Diophantine Problems for Addition and Divisibility

Decidability issues. In general, deciding whether a system of diophantine inequalities has a solution is an undecidable problem [11]. Even more surprising is that very "simple" systems of diophantine inequalities are undecidable. There have been lot of research on setting the exact boundary between decidable and undecidable instances of this problem. Decidability proofs follow often a common scenario: one shows that if there is a solution of a system of diophantine inequalities, then there is a solution bounded by some number that can be effectively determined. So deciding whether there is a solution may be reduced to checking whether there is a solution of a certain size.

Minimal solutions for polynomials of degree 1. In this case, the system consists of a number of linear equations and inequalities.

Theorem 3. ***([4], a restricted version)*** *Let A, b, C, d be respectively $m \times n$, $m \times 1$, $p \times n$, $p \times 1$ matrices with integer coefficients with absolute values bounded by a constant f. If there exists an integer solution x for $Ax = b$ and $Cx \geq d$, then there is a solution x' with absolute values of coefficients bounded by $2^{c_f n}$, where c_f is a constant depending on f.*

Minimal solutions for divisibility. Now consider a system ϕ of divisibility relations

$$f_i(\boldsymbol{x})|g_i(\boldsymbol{x}), \quad \text{for } i \leq K, \tag{1}$$

where each f_i and g_i is a polynomial of degree 1 and variables from x are over $\mathbb{N}$. Recall that an equality $f(\boldsymbol{x}) = g(\boldsymbol{x})$ may be expressed as divisibilities: $f(\boldsymbol{x})|g(\boldsymbol{x})$ and $g(\boldsymbol{x})|f(\boldsymbol{x})$, $f(\boldsymbol{x})+1|g(\boldsymbol{x})+1$, $g(\boldsymbol{x})+1|f(\boldsymbol{x})+1$. An inequality $f(x) > g(x)$ may by expressed by an equality $f(x)+y+1 = g(x)$ with a new variable y, and then in turn by divisibilities.

In papers [9,10], Lipshitz discusses the problem of existence of natural solutions of systems (1). In [9], he shows that the problem is decidable by reducing it to the question of existence of solutions modulo p^s for certain prime numbers p and $s \in \mathbb{N}$ which depends on the system. In [10], he argues that for each fixed K, the minimal solutions, if exist, are of a polynomial size. Note that from the example of language L_2 it follows that the degree of this polynomial grows with K.

For our purposes, we follow the arguments from [9] and state some properties implicitly contained there. The first step in Lipshitz's construction is to replace a system of diophantine divisibilities by systems of totally positive diophantine divisibilities (for a technical definition see [9]). This special form of diophantine divisibilities is crucial for finding "small" solutions.

Theorem 4. ***([9], Section 1)*** *For each system of diophantine divisibilities ϕ there is a set $\phi_1, \ldots, \phi_m$ of totally positive systems of diophantine divisibilities, so that the set of solutions to ϕ coincides with the union of the sets of solutions for $\phi_1, \ldots, \phi_m$.*

In the following parts we shall assume that ϕ is a system of diophantine divisibilities with all coefficients bounded by some constant. Let N be an upper bound on the number of variables and on the number of divisibilities occurring in ϕ. By following the original proof, we get an estimation on the size of coefficients:

Proposition 1. *There is a polynomial w such that absolute value of each coefficient occurring in any ϕ_i is bounded by $2^{2^{w(N)}}$.*

The next step in the construction of Lipshitz is to find a "small" solution for ϕ_i provided that we know that a solution does exist. This is given by the next two theorems:

Theorem 5. ***([9], Lemma 5)*** *If ψ is a system of totally positive diophantine divisibilities, p is a prime number, and there is a solution to ψ, then there is a solution $\bmod p^{k_p}$, such that for every divisor $f_i(x)$ from ψ is different from $0 \bmod p^{k_p}$. The number k_p can be bounded in terms of the maximum value of coefficients in ψ, the number of divisibilities and the number of variables.*

Proposition 2. *If we start with a ϕ as above, then p^{k_p} is bounded by $2^{2^{2^{v(N)}}}$ for some polynomial v.*

Theorem 6. ***([9], Lemma 6)*** *Let ψ be a system of totally positive diophantine divisibilities. There is $M_\phi \in \mathbb{N}$ depending on the maximum value of coefficients, the number of divisibilities and the number of variables in ψ such that the following property holds. If for every prime number $p < M_\phi$, there is a solution of ψ modulo p^{k_p} such that every divisor $f_i(x)$ in ψ holds $f_i(x) \neq 0 \bmod p^{k_p}$, then there is an integer solution to ψ.*

Again, following the construction we may check the following property:

Proposition 3. *The integer solution found in Theorem 6 for the system ϕ is bounded by $2^{2^{2^{u(N)}}}$, where u is a polynomial.*

Conclusions. Still, there is a gap between the bound $\Theta(\log\log n)$ given by L_2 and the lower bound of Theorem 2. To close it, one may try to sharpen the results concerning minimal solutions of systems of diophantine divisibilities (compare also [10] and [2]).

Acknowledgment. We would like thank Krzysztof Loryś for very helpful discussions.

References

1. P. Beame, M. Tompa, P. Yan, *Communication-Space Tradeoffs for Unrestricted Protocols*, SICOMP 23 (1994), 652–661.
2. P. J. Cohen, *Decision procedures for real and p-adic fields*, Comm. on Pure and Applied Math. 22 (1969), 131–151.
3. E. Contejean, H. Devie, *An efficient algorithm for solving systems of diophantine equations*, Information and Computation 113 (1994), 143–172.
4. J. von zur Gathen, M. Sieveking, *A bound on solutions of linear integer equalities and inequalities*, Proc. of the AMS 72(1) (1978), 155–158.
5. M. Holzer, *Multi-Head Finite Automata: Data-Independent Versus Data-Dependent Computations*, Proc. MFCS'97, LNCS 1295, Springer Verlag, Berlin, 1997, 299–309.
6. J. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.
7. T. Jurdziński, *Communication Aspects of Computation of Systems of Finite Automata*, Wrocław University, 2000.
(`http://www.ii.uni.wroc.pl/~acm/doktoraty.html`)
8. T. Jurdziński, M. Kutyłowski, K. Loryś, *Multiparty finite computations*, in Computing and Combinatorics, Proc. COCOON'99, LNCS 1627, Springer Verlag, Berlin, 1999, 318–329.
9. L. Lipshitz, *The diophantine problem for addition and divisibility*, J. AMS 235 (1978), 271–283.
10. L. Lipshitz, *Some remarks on the diophantine problem for addition and divisibility*, Bull. Soc. Math Belg. 33 (1981), 41–52.
11. Ju. Matijasevič, *Hilbert's tenth problem*, Foundations in Computing Series, MIT Press, Cambridge, 1993.
12. V. Mitrana, *On the degree of communication in parallel communicating finite automata systems*, Journal of Automata, Languages and Computation Vol. 5, 3(2000), 301–314.

Separating Quantum and Classical Learning

Rocco A. Servedio

Division of Engineering and Applied Sciences, Harvard University
Cambridge, MA 02138
rocco@deas.harvard.edu

Abstract. We consider a model of learning Boolean functions from *quantum membership queries*. This model was studied in [26], where it was shown that any class of Boolean functions which is information-theoretically learnable from polynomially many quantum membership queries is also information-theoretically learnable from polynomially many classical membership queries.
In this paper we establish a strong *computational* separation between quantum and classical learning. We prove that if any cryptographic one-way function exists, then there is a class of Boolean functions which is polynomial-time learnable from quantum membership queries but not polynomial-time learnable from classical membership queries. A novel consequence of our result is a quantum algorithm that breaks a general cryptographic construction which is secure in the classical setting.

1 Introduction

Over the past decade the study of quantum computation has generated much excitement and attracted intense research attention. One of the most interesting aspects of this new computing paradigm is the possibility that polynomial-time quantum computation may be strictly more powerful than polynomial-time classical computation, i.e. that the quantum class BQP may strictly contain BPP. Evidence for this possibility has been provided by Shor [27], who gave polynomial-time quantum algorithms for factoring and discrete logarithms, two problems which are not known to have polynomial-time classical algorithms.

Since many important learning problems are not known to be solvable in polynomial time, from a learning theory perspective the prospect of gaining computing power via quantum computation is quite intriguing. It is natural to ask whether efficient quantum algorithms can be designed for learning problems (such as the problem of learning DNF formulae) which have thus far resisted efforts to construct polynomial-time algorithms. The ultimate goal of research along these lines would be to construct quantum algorithms which learn using traditional (classical) example oracles, but such algorithms are not yet known.

1.1 Previous Work

As a first step in this direction, several researchers have studied quantum learning algorithms which have access to *quantum oracles*; so in this framework the source

F. Orejas, P.G. Spirakis, and J. van Leeuwen (Eds.): ICALP 2001, LNCS 2076, pp. 1065–1080, 2001.

of examples as well as the learning algorithm itself is assumed to operate in a quantum fashion. The first work along these lines is due to Bshouty and Jackson [10] who defined a quantum PAC oracle and gave an efficient algorithm for learning DNF from a uniform-distribution quantum PAC oracle.

In a different community, many complexity theory researchers have studied the power of quantum computation with a black-box quantum oracle [3,4,5,6,7,8, 11,13,16,28,31]. This research has focused on understanding the relationship between the number of quantum versus classical black-box oracle queries required to determine whether or not a black-box function has some particular property such as ever taking a nonzero value [4,6,11,16,31] or being evenly balanced between the outputs zero and one [13].

Since a classical black-box oracle query to a Boolean function is the same thing as a membership query in learning theory, in light of the work described above it is natural to consider a model of *learning from quantum membership queries*. In contrast with the work described above, the goal here is to *exactly identify* the black-box function rather than to determine whether or not it has some property. Servedio and Gortler [26] defined such a model and proved that any concept class C which is information-theoretically learnable from polynomially many quantum membership queries is also information-theoretically learnable from polynomially many classical membership queries (they also gave a similar result for the quantum PAC learning model of [10]).

This result from [26] deals only with query complexity and does not address the *computational* complexity of quantum versus classical learning. Indeed, [26] also showed that polynomial-time quantum learning is more powerful than polynomial-time classical learning under the assumption that factoring is computationally hard for classical computers. This follows directly from the observation that Shor's quantum factoring algorithm enables quantum algorithms to efficiently learn concept classes whose classical learnability is directly related to the hardness of factoring [2,20].

1.2 Our Results

We give strong evidence that efficient quantum learning algorithms are more powerful than efficient classical learning algorithms. Our main result is the following theorem:

Theorem 1. *If any one-way function exists, then there is a concept class C which is polynomial-time learnable from quantum membership queries but is not polynomial-time learnable from classical membership queries.*

This separation between quantum and classical learning is far more robust than previous work [26]. Even if a polynomial-time classical factoring algorithm were to be discovered, our separation would hold as long as *any* one-way function exists (a universally held belief in public-key cryptography).

The main cryptographic tool underlying our results is a new construction of pseudorandom functions which are invariant under an XOR mask (see Section

4). As described in Section 5.1, each concept $c \in C$ combines these new pseudorandom functions with pseudorandom permutations in a particular way. Roughly speaking, the XOR mask invariance of the new pseudorandom functions ensures that a quantum algorithm due to Simon [28] can be used to extract some information about the structure of the target concept and thus make progress towards learning. On the other hand, the pseudorandomness ensures that no probabilistic polynomial-time learning algorithm can extract any useful information, and thus no such algorithm can learn successfully.

As discussed in Section 6, our results prove the existence of a quantum oracle algorithm which defeats a general cryptographic construction secure in the classical setting. More precisely, given any one-way function we construct a family of pseudorandom functions which are classically secure but can be distinguished from truly random functions (and in fact exactly identified) by an algorithm which makes quantum oracle queries. To our knowledge, this is the first break of a general cryptographic protocol (not based on a specific assumption such as factoring) in a quantum setting.

2 Preliminaries

A *concept* is a Boolean function $c : \{0,1\}^n \to \{0,1\}$. A *concept class* $C = \cup_{n \geq 1} C_n$ is a collection of concepts with $C_n = \{c \in C : \ c \text{ is a concept over } \{0,1\}^n\}$.

For $\alpha, \beta \in \{0,1\}$ we write $\alpha \oplus \beta$ to denote the exclusive-or $\alpha + \beta \pmod 2$. Similarly for $x, y \in \{0,1\}^n$ we write $x \oplus y$ to denote the n-bit string which is the bitwise XOR of x and y. We write $x \cdot y$ to denote the inner product $x_1 y_1 + \cdots + x_n y_n \pmod 2$, and we write $|x|$ to denote the length of string x.

We use script capital letters to denote probability distributions over sets of functions; in particular $\mathcal{F}_n$ denotes the uniform distribution over all 2^{n2^n} functions from $\{0,1\}^n$ to $\{0,1\}^n$. If S is a finite set we write $\Pr_{s \in S}$ to denote a uniform choice of s from S.

We write $M(s)$ to indicate that algorithm M is given string s as input and M^g to indicate that M has access to an oracle for the function g. If M is a probabilistic polynomial-time (henceforth abbreviated p.p.t.) algorithm which has access to an oracle $g : \{0,1\}^{\ell_1} \to \{0,1\}^{\ell_2}$, then the running time of M^g is bounded by $p(\ell_1 + \ell_2)$ for some polynomial p.

In the model of *exact learning from membership queries* [1,9,14,18,19], the learning algorithm is given access to an oracle for an unknown *target concept* $c \in C_n$. When invoked on input $x \in \{0,1\}^n$, the oracle returns $c(x)$ (such an oracle call is known as a *membership query*). The goal of the learning algorithm is to construct a hypothesis $h : \{0,1\}^n \to \{0,1\}$ which is logically equivalent to c, i.e. $h(x) = c(x)$ for all $x \in \{0,1\}^n$. We say that a concept class C is *polynomial-time learnable from membership queries* if there is a p.p.t. oracle algorithm A with the following property: for all $n \geq 1$, for all $c \in C_n$, with probability at least $9/10$ A^c outputs a Boolean circuit h which is equivalent to c.

Detailed descriptions of quantum Turing machine and circuit models and of quantum oracle computation are given in [5,12,25,30]. Most of the details are irrelevant for our needs, so we describe here only the essential aspects.

A quantum computation takes place over an m-bit *quantum register*. At any stage in the execution of a quantum computation the state of this register is some superposition $\sum_{z\in\{0,1\}^m} \alpha_z|z\rangle$ of basis states $|z\rangle$, where the α_z are complex numbers which satisfy the constraint $\sum_{z\in\{0,1\}^m} \|\alpha_z\|^2 = 1$. We may thus view the state of a quantum register as being a 2^m-dimensional complex vector of unit norm. A single step of a quantum computation corresponds to a unitary transformation of this vector, i.e. a $2^m \times 2^m$ unitary matrix, and a quantum algorithm is defined by a sequence of such matrices. A single quantum computation step is analogous to a single gate in a classical circuit; in order to ensure that each step of quantum computation performs only a bounded amount of work, the model stipulates that each unitary matrix must be simple and local in a well-defined sense. At the end of a quantum computation a *measurement* is performed on the register and an m-bit string is obtained as output. If the register's final state is $\sum_{z\in\{0,1\}^m} \alpha_z|z\rangle$ then with probability $\|\alpha_z\|^2$ the string $z \in \{0,1\}^m$ is the output.

Quantum computation with access to an oracle $c : \{0,1\}^n \to \{0,1\}^\ell$ proceeds as follows. If the oracle c is invoked when the current state of the quantum register is $\sum_{x\in\{0,1\}^n, y\in\{0,1\}^\ell, w\in\{0,1\}^{m-n-\ell}} \alpha_{x,y,w}|x,y,w\rangle$, then at the next step the state of the register will be $\sum_{x\in\{0,1\}^n, y\in\{0,1\}^\ell, w\in\{0,1\}^{m-n-\ell}} \alpha_{x,y,w}|x,y\oplus c(x),w\rangle$. (It can be verified that this change of state is a unitary transformation for any oracle c.) Thus a quantum algorithm can invoke its oracle on a superposition of inputs and receives a corresponding superposition of responses. This model of quantum computation with an oracle has been studied by many researchers in complexity theory [3,4,5,6,7,8,11,13,16,28,31] and has also recently been studied from a learning theory perspective [26].

We say that a concept class C is *polynomial-time learnable from quantum membership queries* if there is a quantum oracle algorithm A with the following property: for all $n \geq 1$, for all $c \in C_n$, A^c runs for at most $\mathrm{poly}(n)$ steps and with probability at least $9/10$ outputs a representation of a Boolean circuit h which is logically equivalent to c. It is well known that any classical oracle algorithm can be efficiently simulated by a quantum oracle algorithm, and thus any concept class which is polynomial-time learnable from classical membership queries is polynomial-time learnable from quantum membership queries.

3 Simon's Algorithm

Let $f : \{0,1\}^n \to \{0,1\}^n$ be a function and let $0^n \neq s \in \{0,1\}^n$. We say that f is *two-to-one with XOR mask* s if for all $y \neq x$, $f(x) = f(y) \iff y = x\oplus s$. More generally, f is *invariant under XOR mask with* s if $f(x) = f(x\oplus s)$ for all $x \in \{0,1\}^n$ (note that such a function need not be two-to-one).

Simon [28] has given a simple quantum algorithm which takes oracle access to a function $f : \{0,1\}^n \to \{0,1\}^n$, runs in $\mathrm{poly}(n)$ time, and behaves as follows:

1. If f is a permutation on $\{0,1\}^n$, the algorithm outputs an n-bit string y which is uniformly distributed over $\{0,1\}^n$.
2. If f is two-to-one with XOR mask s, the algorithm outputs an n-bit string y which is uniformly distributed over the 2^{n-1} strings such that $y \cdot s = 0$.
3. If f is invariant under XOR mask with s, the algorithm outputs some n-bit string y which satisfies $y \cdot s = 0$.

Simon showed that by running this procedure $O(n)$ times a quantum algorithm can distinguish between Case 1 (f is a permutation) and Case 3 (f is invariant under some XOR mask) with high probability. In Case 1 after $O(n)$ repetitions the strings obtained will with probability $1 - 2^{-O(n)}$ contain a basis for the vector space $(Z_2)^n$ (here we are viewing n-bit strings as vectors over Z_2), while in Case 3 the strings obtained cannot contain such a basis since each string must lie in the subspace $\{y : y \cdot s = 0\}$. Simon also observed that in Case 2 (f is two-to-one with XOR mask s) the algorithm can be used to efficiently identify s with high probability. This is because after $O(n)$ repetitions, with high probability s will be the unique nonzero vector whose dot product with each y is 0; this vector can be found by solving the linear system defined by the y's.

Simon also analyzed the success probability of classical oracle algorithms for this problem. His analysis establishes the following theorem:

Theorem 2. *Let $0^n \neq s \in \{0,1\}^n$ be chosen uniformly and let $f : \{0,1\}^n \to \{0,1\}^n$ be an oracle chosen uniformly from the set of all functions which are two-to-one with XOR mask s. Then (i) there is a polynomial-time quantum oracle algorithm which identifies s with high probability; (ii) any p.p.t. classical oracle algorithm identifies s with probability $1/2^{\Omega(n)}$.*

This surprising ability of quantum oracle algorithms to efficiently find s is highly suggestive in the context of our quest for a learning problem which separates polynomial-time classical and quantum computation. Indeed, Simon's algorithm will play a crucial role in establishing that the concept class which we construct in Section 5 is learnable in poly(n) time by a quantum algorithm. Recall that in our learning scenario, though, the goal is to *exactly identify* the unknown target function, not just to identify the string s. Since $2^{\Omega(n)}$ bits are required to specify a randomly chosen function f which is two-to-one with XOR mask s, no algorithm (classical or quantum) can output a description of f in poly(n) time, much less learn f in poly(n) time. Thus it will not do to use truly random functions for our learning problem; instead we use *pseudorandom* functions as described in the next section.

4 Pseudorandomness

A *pseudorandom function family* [15] is a collection of functions $\{f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s \in \{0,1\}^*}$ with the following two properties: (i) *(efficient evaluation)* there is a deterministic algorithm which, given an n-bit seed s and an n-bit input x,

runs in time poly(n) and outputs $f_s(x)$; (ii) *(pseudorandomness)* for all polynomials Q, all p.p.t. oracle algorithms M, and all sufficiently large n, we have that $\left|\Pr_{F\in\mathcal{F}_n}[M^F \text{ outputs } 1] - \Pr_{s\in\{0,1\}^n}[M^{f_s} \text{ outputs } 1]\right| < \frac{1}{Q(n)}$. Intuitively, the pseudorandomess property ensures that in any p.p.t. computation which uses a truly random function, a randomly chosen pseudorandom function may be used instead without affecting the outcome in a noticeable way. Well known results [15,17] imply that pseudorandom function families exist if and only if any one-way function exists.

A *pseudorandom permutation family* is a pseudorandom function family with the added property that each function $f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}$ is a permutation. Luby and Rackoff [21] gave the first construction of a pseudorandom permutation family from any pseudorandom function family. In their construction each permutation $f_s : \{0,1\}^n \to \{0,1\}^n$ has a seed s of length $|s| = 3n/2$ rather than n as in our definition above. Subsequent constructions [22,23,24] of pseudorandom permutation families $\{f_s : \{0,1\}^n \to \{0,1\}^n\}$ use n-bit seeds and hence match our definition exactly. (Our definition of pseudorandomness could easily be extended to allow seed lengths other than n. For our construction in Section 5 it will be convenient to have n-bit seeds.)

4.1 Pseudorandom Functions Invariant under XOR Mask

Our main cryptographic result, stated below, is proved in Appendix A:

Theorem 3. *If any one-way function exists, then there is a pseudorandom function family* $\{g_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}$ *such that* $g_s(x) = g_s(x \oplus s)$ *for all* $|x| = |s|$.

A first approach to constructing such a family is as follows: given any pseudorandom function family $\{f_s\}$, let $\{g_s\}$ be defined by

$$g_s(x) \stackrel{\text{def}}{=} f_s(x) \oplus f_s(x \oplus s). \tag{1}$$

This simple construction ensures that each function g_s is invariant under XOR mask with s, but the family $\{g_s\}$ need not be pseudorandom just because $\{f_s\}$ is pseudorandom. Indeed, if $\{h_s\}$ is similarly defined by $h_s(x) \stackrel{\text{def}}{=} g_s(x) \oplus g_s(x \oplus s)$, then $\{h_s\}$ is not pseudorandom since $h_s(x) = (f_s(x) \oplus f_s(x \oplus s)) \oplus (f_s(x \oplus s) \oplus f_s(x \oplus s \oplus s)) = 0^n$.

While this example shows that (1) does not always preserve pseudorandomness, it leaves open the possibility that (1) may preserve pseudorandomness for certain function families $\{f_s\}$. In Appendix A we show that if $\{f_s\}$ is a pseudorandom function family which is constructed from any one-way function in a particular way, then the family $\{g_s\}$ defined by (1) is indeed pseudorandom.

It may at first appear that the pseudorandom function family $\{g_s\}$ given by Theorem 3 immediately yields a concept class which separates efficient quantum learning from efficient classical learning. The pseudorandomness of $\{g_s\}$ ensures that no p.p.t. algorithm can learn successfully; on the other hand, if Simon's quantum algorithm is given oracle access to a function which is two-to-one with

XOR mask s, then it can efficiently find s with high probability. Hence it may seem that given access to g_s Simon's quantum algorithm can efficiently identify the seed s and thus learn the target concept.

The flaw in this argument is that each function g_s from Theorem 3, while invariant under XOR mask with s, need not be two-to-one. Indeed g_s could conceivably be invariant under XOR mask with, say, $\sqrt{n}$ linearly independent strings $s = s^1, s^2, \ldots, s^{\sqrt{n}}$. Such a set of strings spans a $2^{\sqrt{n}}$-element subspace of $\{0,1\}^n$; even if Simon's algorithm could identify this subspace, it would not indicate which element of the subspace is the true seed s. Hence a more sophisticated construction is required.

5 Proof of Theorem 1

5.1 The Concept Class C

We describe concepts over $\{0,1\}^m$ where $m = n + 2\log n + 1$. Each concept in C_m is defined by an $(n+1)$-tuple $(y, s^1, \ldots, s^n)$ where $y = y_1 \ldots y_n \in \{0,1\}^n$ and each $s^i \in \{0,1\}^n \setminus \{0^n\}$, so C_m contains $2^n(2^n-1)^n$ distinct concepts. For brevity we write $\overline{s}$ to stand for $s^1, \ldots, s^n$ below.

Roughly speaking, each concept in C_m comprises n pseudorandom functions; as explained below the string y acts as a "password" and the strings $s^1, \ldots, s^n$ are the seeds to the pseudorandom functions. Each concept $c \in C_m$ takes m-bit strings as inputs; we view such an m-bit input as a 4-tuple (b, x, i, j) where $b \in \{0,1\}$, $x \in \{0,1\}^n$, and $i, j \in \{0,1\}^{\log n}$ each represent a number in the range $\{1, 2, \ldots, n\}$.

Let $\{h_s^0 : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s \in \{0,1\}^*}$ be a pseudorandom permutation family and let $\{h_s^1 : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s \in \{0,1\}^*}$ be the pseudorandom function family from Theorem 3, so $h_s^1(x) = h_s^1(x \oplus s)$. The concept $c_{y,\overline{s}}$ is defined as follows on input (b, x, i, j) :

- **If $b = 0$**: A query $(0, x, i, j)$ is called a *function query*. The value of $c_{y,\overline{s}}(0, x, i, j)$ is $h_{s^i}^{y_i}(x)_j$, i.e. the j-th bit of the n-bit string $h_{s^i}^{y_i}(x)$. Thus the bit y_i determines whether the i-th pseudorandom function used is a permutation or is invariant under XOR mask with s^i.
- **If $b = 1$**: A query $(1, x, i, j)$ is called a *seed query*. The value of $c_{y,\overline{s}}(1, x, i, j)$ is 0 if $x \neq y$ and is s_j^i (the j-th bit of the i-th seed s^i) if $x = y$.

The intuition behind our construction is simple: in order to learn the target concept successfully a learning algorithm must identify each seed string $s^1, \ldots, s^n$. These strings can be identified by making seed queries $(1, y, i, j)$, but in order to make the correct seed queries the learning algorithm must know y. Since each bit y_i corresponds to whether an oracle is a permutation or is XOR-mask invariant, a quantum algorithm can determine each y_i and thus can learn successfully. However, no p.p.t. algorithm can distinguish between these two types of oracles (since in either case the oracle is pseudorandom and hence is indistinguishable from a truly random function), so no p.p.t. algorithm can learn y.

5.2 A Quantum Algorithm Which Learns C in Polynomial Time

Theorem 4. *C is polynomial-time learnable from quantum membership queries.*

Proof sketch: Let $c_{y,\bar{s}} \in C_m$ be the target concept. Each function $h_{s^i}^{y_i}$ is a permutation iff $y_i = 0$ and is XOR-mask invariant iff $y_i = 1$ (this is why we do not allow $s^i = 0^n$ in the definition of the concept class). Using quantum membership queries, a poly(n)-time quantum algorithm can run Simon's procedure n times, once for each function $h_{s^i}^{y_i}$, and thus determine each bit y_i with high probability. (One detail which arises here is that Simon's algorithm uses an oracle $\{0,1\}^n \to \{0,1\}^n$ whereas in our learning setting the oracle outputs one bit at a time. This is not a problem since it is possible to simulate any call to Simon's oracle by making n sequential calls, bit by bit, to our oracle.) Given the string $y = y_1 \dots y_n$, the algorithm can then make n^2 queries on inputs $(1, y, i, j)$ for $1 \le i, j \le n$ to learn each of the n strings $s^1, \dots, s^n$. Once y and $s^1, \dots, s^n$ are known it is straightforward to output a circuit for $c_{y,\bar{s}}$. □

5.3 No Classical Algorithm Learns C in Polynomial Time

Theorem 5. *C is not polynomial-time learnable from classical membership queries.*

Let $C'_m \supset C_m$, $|C'_m| = 2^{n^2+n}$ be the concept class $C'_m = \{c_{y,\bar{s}} : y, s^1, \dots, s^n \in \{0,1\}^n\}$; thus C'_m includes concepts in which s^i may be 0^n. The following lemma states that it is hard to learn a target concept chosen uniformly from C'_m :

Lemma 1. *For all polynomials Q, all p.p.t. learning algorithms A, and all sufficiently large n,* $\Pr_{c_{y,\bar{s}} \in C'_m}[A^{c_{y,\bar{s}}}$ *outputs a hypothesis* $h \equiv c_{y,\bar{s}}] < \frac{1}{Q(n)}$.

To see that Lemma 1 implies Theorem 5, we note that the uniform distribution over C'_m and the uniform distribution over C_m are nearly identical (the two distributions have total variation distance $O(n/2^n)$). Lemma 1 thus has the following analogue for C_m which clearly implies Theorem 5:

Lemma 2. *For all polynomials Q, all p.p.t. learning algorithms A, and all sufficiently large n,* $\Pr_{c_{y,\bar{s}} \in C_m}[A^{c_{y,\bar{s}}}$ *outputs a hypothesis* $h \equiv c_{y,\bar{s}}] < \frac{1}{Q(n)}$.

The proof of Lemma 1 proceeds as follows: we say that a learning algorithm A *hits* y if at some point during its execution A makes a seed query $(1, y, i, j)$, and we say that A *misses* y if A does not hit y. We have that

$$\begin{aligned}
\Pr_{c_{y,\bar{s}} \in C'_m}[A^{c_{y,\bar{s}}} \text{ outputs } h \equiv c_{y,\bar{s}}] &= \Pr[A^{c_{y,\bar{s}}} \text{ outputs } h \equiv c_{y,\bar{s}} \ \ \& \ \ A^{c_{y,\bar{s}}} \text{ hits } y] + \\
&\quad \Pr[A^{c_{y,\bar{s}}} \text{ outputs } h \equiv c_{y,\bar{s}} \ \ \& \ \ A^{c_{y,\bar{s}}} \text{ misses } y] \\
&\le \Pr[A^{c_{y,\bar{s}}} \text{ hits } y] + \\
&\quad \Pr[A^{c_{y,\bar{s}}} \text{ outputs } h \equiv c_{y,\bar{s}} \ \mid \ A^{c_{y,\bar{s}}} \text{ misses } y].
\end{aligned}$$

Lemma 1 thus follows from the following two lemmas:

Lemma 3. *For all polynomials Q, all p.p.t. learning algorithms A, and all sufficiently large n,* $\Pr_{c_{y,\bar{s}} \in C'_m}[A^{c_{y,\bar{s}}} \text{ hits } y] < \frac{1}{Q(n)}$.

Lemma 4. *For all polynomials Q, all p.p.t. learning algorithms A, and all sufficiently large n,* $\Pr_{c_{y,\bar{s}} \in C'_m}[A^{c_{y,\bar{s}}} \text{ outputs } h \equiv c_{y,\bar{s}} \mid A^{c_{y,\bar{s}}} \text{ misses } y] < \frac{1}{Q(n)}$.

Proof of Lemma 3. The idea of the proof is as follows: before hitting y for the first time algorithm A gets 0 as the answer to each seed query, so A might as well be querying a modified oracle which answers 0 to *every* seed query. We show that no p.p.t. algorithm which has access to such an oracle can output y with inverse polynomial success probability (intuitively this is because such an oracle consists entirely of pseudorandom functions and hence can provide no information to any p.p.t. algorithm), and thus A's probability of hitting y must be less than $1/\text{poly}(n)$ as well.

More formally, let A be any p.p.t. learning algorithm. Without loss of generality we may suppose that A always makes exactly $q(n)$ seed queries during its execution for some polynomial q. Let $X^1, \ldots, X^{q(n)}$ be the sequence of strings in $\{0,1\}^n$ on which $A^{c_{y,\bar{s}}}$ makes its seed queries, i.e. $A^{c_{y,\bar{s}}}$ uses $(1, X^t, i_t, j_t)$ as its t-th seed query. Each X^t is a random variable over the probability space defined by the uniform choice of $c_{y,\bar{s}} \in C'_m$ and any internal randomness of algorithm A.

For each $c_{y,\bar{s}} \in C'_m$ let $\tilde{c}_{y,\bar{s}} : \{0,1\}^m \to \{0,1\}$ be a modified version of $c_{y,\bar{s}}$ which answers 0 to all seed queries, i.e. $\tilde{c}_{y,\bar{s}}(b,x,i,j)$ is $c_{y,\bar{s}}(b,x,i,j)$ if $b = 0$ and is 0 if $b = 1$. Consider the following algorithm B which takes access to an oracle for $\tilde{c}_{y,\bar{s}}$ and outputs an n-bit string. B executes algorithm $A^{\tilde{c}_{y,\bar{s}}}$ (note that the oracle used is $\tilde{c}_{y,\bar{s}}$ rather than $c_{y,\bar{s}}$), then chooses a uniform random value $1 \le t \le q(n)$ and outputs $\tilde{X}^t$, the string on which $A^{\tilde{c}_{y,\bar{s}}}$ made its t-th seed query. Like the X^ts, each $\tilde{X}^t$ is a random variable over the probability space defined by a uniform choice of $c_{y,\bar{s}} \in C'_m$ and any internal randomness of A.

The following two lemmas together imply Lemma 3:

Lemma 5. $2q(n)^2 \cdot \Pr_{c_{y,\bar{s}} \in C'_m}[B^{\tilde{c}_{y,\bar{s}}} \text{ outputs } y] \ge \Pr_{c_{y,\bar{s}} \in C'_m}[A^{c_{y,\bar{s}}} \text{ hits } y]$.

Lemma 6. $\left|\Pr_{c_{y,\bar{s}} \in C'_m}[B^{\tilde{c}_{y,\bar{s}}} \text{ outputs } y] - \frac{1}{2^n}\right| < \frac{1}{Q(n)}$ *for all polynomials Q and all sufficiently large n.*

Proof of Lemma 5. We have that

$$\sum_{t=1}^{q(n)} \Pr[X^t = y \ \ \& \ \ X^\tau \neq y \text{ for } \tau < t] \le \sum_{t=1}^{q(n)} \Pr[X^t = y \mid X^\tau \neq y \text{ for } \tau < t].$$

Since the left side of this inequality is exactly $\Pr_{c_{y,\bar{s}} \in C'_m}[A^{c_{y,\bar{s}}} \text{ hits } y]$, for some value $1 \le t_0 \le q(n)$ we have

$$\Pr[X^{t_0} = y \mid X^\tau \neq y \text{ for } \tau < t_0] \ \ge \ \Pr[A^{c_{y,\bar{s}}} \text{ hits } y]/q(n). \tag{2}$$

Since the distribution of responses to function queries which A makes prior to its first seed query is the same regardless of whether the oracle is $c_{y,\bar{s}}$ or $\tilde{c}_{y,\bar{s}}$, it is clear that the random variables X^1 and $\tilde{X}^1$ are identically distributed. An inductive argument shows that for all $t \geq 1$, the conditional random variables $X^t \mid (X^\tau \neq y \text{ for } \tau < t)$ and $\tilde{X}^t \mid (\tilde{X}^\tau \neq y \text{ for } \tau < t)$ are identically distributed (in each case the conditioning ensures that the distribution of responses to seed queries which A makes prior to its t-th seed query is the same, i.e. all 0).

We consider two possible cases. If $\Pr_{c_{y,\bar{s}} \in C'_m}[\tilde{X}^\tau \neq y \text{ for } \tau < t_0] > 1/2$, then

$$\begin{aligned}
\Pr_{c_{y,\bar{s}} \in C'_m}[B^{\tilde{c}_{y,\bar{s}}} \text{ outputs } y] &\geq \Pr[B^{\tilde{c}_{y,\bar{s}}} \text{ chooses } t_0] \cdot \Pr[\tilde{X}^{t_0} = y \ \&\ \tilde{X}^\tau \neq y \text{ for } \tau < t_0] \\
&= \frac{\Pr[\tilde{X}^{t_0} = y \mid \tilde{X}^\tau \neq y \text{ for } \tau < t_0] \cdot \Pr[\tilde{X}^\tau \neq y \text{ for } \tau < t_0]}{q(n)} \\
&> \Pr[\tilde{X}^{t_0} = y \mid \tilde{X}^\tau \neq y \text{ for } \tau < t_0]/2q(n) \\
&= \Pr[X^{t_0} = y \mid X^\tau \neq y \text{ for } \tau < t_0]/2q(n) \\
&\geq \Pr[A^{c_{y,\bar{s}}} \text{ hits } y]/2q(n)^2. \qquad \text{by (2)}
\end{aligned}$$

Otherwise if $\Pr_{c_{y,\bar{s}} \in C'_m}[\tilde{X}^\tau \neq y \text{ for } \ \tau < t_0] \leq 1/2$, then $\sum_{t=0}^{t_0-1} \Pr[\tilde{X}^t = y] \geq 1/2$ and hence $\Pr_{c_{y,\bar{s}} \in C'_m}[B^{\tilde{c}_{y,\bar{s}}} \text{ outputs } y]$ is at least

$$\sum_{t=1}^{t_0-1} \Pr[B^{\tilde{c}_{y,\bar{s}}} \text{ chooses } t] \cdot \Pr[\tilde{X}^t = y] \geq \frac{1}{2q(n)} \geq \frac{\Pr[A^{c_{y,\bar{s}}} \text{ hits } y]}{2q(n)^2}. \qquad \square$$

Proof of Lemma 6. For $z, \zeta \in \{0,1\}^n$ let $p^z_\zeta = \Pr_{c_{y,\bar{s}} \in C'_m}[B^{\tilde{c}_{y,\bar{s}}} \text{ outputs } z \mid y = \zeta]$. For $\ell \in \{1, \ldots, n\}$ let $\zeta || \ell$ denote ζ with the ℓ-th bit flipped. Similarly, for $S \subseteq \{1, \ldots, n\}$ let $\zeta || S$ denote ζ with bits flipped in all positions corresponding to S.

Fix $z, \zeta \in \{0,1\}^n$ and $\ell \in \{1, \ldots, n\}$ and consider the following algorithm $D_{z,\zeta,\ell}$ which takes access to an oracle $f : \{0,1\}^n \to \{0,1\}^n$ and outputs a single bit: For all $i \neq \ell$ algorithm $D_{z,\zeta,\ell}$ first chooses a random n-bit string s^i. $D_{z,\zeta,\ell}$ then runs algorithm B, simulating the oracle for B as follows:

- queries $(0, x, \ell, j)$ are answered with the bit $f(x)_j$
- for $i \neq \ell$ queries $(0, x, i, j)$ are answered with the bit $h^{\zeta_i}_{s^i}(x)_j$
- all queries $(1, x, i, j)$ are answered with the bit 0.

Finally algorithm $D_{z,\zeta,\ell}$ outputs 1 if B's output is z and outputs 0 otherwise.

It is easy to verify that for all z, ζ, ℓ we have $p^z_\zeta = \Pr_{s \in \{0,1\}^n}[D^{h_s^{\zeta_\ell}}_{z,\zeta,\ell} \text{ outputs } 1]$ and $p^z_{\zeta||\ell} = \Pr_{s \in \{0,1\}^n}[D^{h_s^{1-\zeta_\ell}}_{z,\zeta,\ell} \text{ outputs } 1]$. From the definition of pseudorandomness and the triangle inequality it follows that $|p^z_\zeta - p^z_{\zeta||\ell}| < \frac{1}{nQ(n)}$. Making $|S| \leq n$ applications of this inequality and using the triangle inequality, we find that $|p^z_\zeta - p^z_{\zeta||S}| < \frac{1}{Q(n)}$. We thus have that $|p^z_\zeta - p^z_z| < \frac{1}{Q(n)}$ for all $z, \zeta \in \{0,1\}^n$. Since $\sum_{z \in \{0,1\}^n} p^z_\zeta = 1$, we have that $\left|\Pr_{c_{y,\bar{s}} \in C'_m}[B^{\tilde{c}_{y,\bar{s}}} \text{ outputs } y] - \frac{1}{2^n}\right| = \left|\frac{1}{2^n}\left(\sum_{z \in \{0,1\}^n} p^z_z\right) - \frac{1}{2^n}\right| = \frac{1}{2^n}\left|\sum_{z \in \{0,1\}^n}(p^z_z - p^z_\zeta)\right| < \frac{1}{Q(n)}$. $\square$

Proof of Lemma 4. The idea here is that conditioning on the event that A misses y ensures that the only information which A has about y and $\overline{s}$ comes from querying oracles for the pseudorandom functions $h_{s^i}^{y_i}$. Since these pseudorandom functions are indistinguishable from truly random functions, no p.p.t. algorithm can learn successfully.

Formally, let A be any p.p.t. learning algorithm. Consider the following algorithm B which takes access to an oracle $h_{s^n}^{y_n} : \{0,1\}^n \to \{0,1\}^n$ and outputs a representation of a function $g : \{0,1\}^n \to \{0,1\}^n$. Algorithm B first chooses $\hat{y} = y_1 \dots y_{n-1}$ uniformly from $\{0,1\}^{n-1}$ and chooses $n-1$ strings $s^1, \dots, s^{n-1}$ each uniformly from $\{0,1\}^n$. B then runs algorithm $A^{\tilde{c}_{y,\overline{s}}}$ (observe that B can simulate the oracle $\tilde{c}_{y,\overline{s}}$ since it has access to an oracle for $h_{s^n}^{y_n}$ and knows y_i, s^i for $i \neq n$) which generates some hypothesis h. Finally B outputs the function $g : \{0,1\}^n \to \{0,1\}^n$ defined by $g(x) \stackrel{\text{def}}{=} h(0,x,n,1)h(0,x,n,2)\dots h(0,x,n,n)$.

The following two lemmas together imply Lemma 4:

Lemma 7. *For all sufficiently large n,* $\Pr_{y_n \in \{0,1\}, s^n \in \{0,1\}^n}[B^{h_{s^n}^{y_n}} \text{ outputs } g \equiv h_{s^n}^{y_n}] > \Pr_{c_{y,\overline{s}} \in C'_m}[A^{c_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} \mid A^{c_{y,\overline{s}}} \text{ misses } y]/2$.

Lemma 8. $\Pr_{y_n \in \{0,1\}, s^n \in \{0,1\}^n}[B^{h_{s^n}^{y_n}} \text{ outputs } g \equiv h_{s^n}^{y_n}] < \frac{1}{Q(n)}$ *for all polynomials Q and all sufficiently large n.*

Proof of Lemma 7. It is easy to see that if $A^{\tilde{c}_{y,\overline{s}}}$ outputs a hypothesis which is equivalent to $c_{y,\overline{s}}$, then g will be equivalent to $h_{s^n}^{y_n}$. For sufficiently large n we thus have that $\Pr_{y_n \in \{0,1\}, s^n \in \{0,1\}^n}[B^{h_{s^n}^{y_n}} \text{ outputs } g \equiv h_{s^n}^{y_n}]$ is at least

$$\begin{aligned}
\Pr_{c_{y,\overline{s}} \in C'_m}[A^{\tilde{c}_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}}] &\geq \Pr[A^{\tilde{c}_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} \ \ \& \ \ A^{\tilde{c}_{y,\overline{s}}} \text{ misses } y] \\
&= \Pr[A^{\tilde{c}_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} \ \mid \ A^{\tilde{c}_{y,\overline{s}}} \text{ misses } y] \cdot \\
&\quad \Pr[A^{\tilde{c}_{y,\overline{s}}} \text{ misses } y] \\
&> \Pr[A^{\tilde{c}_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} \ \mid \ A^{\tilde{c}_{y,\overline{s}}} \text{ misses } y]/2
\end{aligned}$$

where the last inequality follows from Lemma 3.

Let $TRANS(A^{c_{y,\overline{s}}})$ ($TRANS(A^{\tilde{c}_{y,\overline{s}}})$ respectively) denote a complete transcript of algorithm A's execution on oracle $c_{y,\overline{s}}$ ($\tilde{c}_{y,\overline{s}}$ respectively). $TRANS(A^{c_{y,\overline{s}}})$ and $TRANS(A^{\tilde{c}_{y,\overline{s}}})$ are each random variables over the probability space defined by a uniform choice of $c_{y,\overline{s}} \in C'_m$ and any internal randomness of algorithm A. An easy induction shows that the two conditional random variables $TRANS(A^{\tilde{c}_{y,\overline{s}}}) \mid (A^{\tilde{c}_{y,\overline{s}}} \text{ misses } y)$ and $TRANS(A^{c_{y,\overline{s}}}) \mid (A^{c_{y,\overline{s}}} \text{ misses } y)$ are identically distributed. This implies that $\Pr_{c_{y,\overline{s}} \in C'_m}[A^{\tilde{c}_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} | A^{\tilde{c}_{y,\overline{s}}} \text{ misses } y] = \Pr_{c_{y,\overline{s}} \in C'_m}[A^{c_{y,\overline{s}}} \text{ outputs } h \equiv c_{y,\overline{s}} | A^{c_{y,\overline{s}}} \text{ misses } y]$, which combined with the inequality above proves the lemma. □

Proof of Lemma 8. The following fact, which follows easily from the pseudorandomness of $\{h^0\}$ and $\{h^1\}$, states that $\{h_s^b\}_{b \in \{0,1\}, s \in \{0,1\}^n}$ is a pseudorandom function family:

Fact 1 *For all polynomials Q, p.p.t. oracle algorithms A, and sufficiently large n, we have* $\left|\Pr_{b\in\{0,1\},s\in\{0,1\}^n}[A^{h_s^b} \text{ outputs } 1] - \Pr_{F\in\mathcal{F}_n}[A^F \text{ outputs } 1]\right| < \frac{1}{Q(n)}$.

Intuitively the pseudorandomness of $\{h_s^b\}$ should make it hard for $B^{h_{s^n}^{y_n}}$ to output $h_{s^n}^{y_n}$ since clearly no p.p.t. algorithm, given oracle access to a truly random function F, could output a function equivalent to F. Formally, we consider an algorithm D which takes oracle access to a function $f : \{0,1\}^n \to \{0,1\}^n$ and outputs a single bit. D runs B^f to obtain a function g and then selects a string $z \in \{0,1\}^n$ which was not used as an oracle query in the computation of B^f. D calls the oracle to obtain $f(z)$, evaluates g to obtain $g(z)$, and ouputs 1 if the two values are equal and 0 otherwise.

Clearly $\Pr[D^f \text{ outputs } 1] \geq \Pr[B^f \text{ outputs } g \equiv f]$. Since $\Pr_{F\in\mathcal{F}_n}[D^F \text{ outputs } 1] = 1/2^n$, using Fact 1 we find that $\left|\Pr_{y_n\in\{0,1\},s^n\in\{0,1\}^n}[D^{h_{s^n}^{y_n}} \text{ outputs } 1] - \frac{1}{2^n}\right| < \frac{1}{2Q(n)}$ and hence $\Pr_{y_n\in\{0,1\},s^n\in\{0,1\}^n}[B^{h_{s^n}^{y_n}} \text{ outputs } g \equiv h_{s^n}^{y_n}] < \frac{1}{Q(n)}$. □

6 Breaking Classical Cryptography in a Quantum Setting

Our constructions highlight some interesting issues concerning the relation between quantum oracle computation and classical cryptography. It is clear that a quantum algorithm, given access to a quantum black-box oracle for an unknown function, can efficiently distinguish between truly random functions and pseudorandom functions drawn from the family $\{g_s\}$ of Theorem 3. Our construction of $\{g_s\}$ thus shows that cryptographic constructions which are provably secure in the classical model can fail in a quantum setting. We emphasize that this failure does *not* depend on the ability of polynomial-time quantum algorithms to invert particular one-way functions such as factoring; even if no quantum algorithm can efficiently invert the one-way function used to construct $\{g_s\}$, our results show that a polynomial-time quantum algorithm can be a successful distinguisher. It would be interesting to obtain stronger constructions of pseudorandom functions which are provably secure in the quantum oracle framework.

Acknowledgements. We thank S. Gortler and A. Klivans for stimulating discussions.

References

1. D. Angluin. Queries and concept learning. *Machine Learning* **2** (1988), 319-342.
2. D. Angluin and M. Kharitonov. When won't membership queries help? *J. Comp. Syst. Sci.* **50** (1995), 336-355.
3. R. Beals, H. Buhrman, R. Cleve, M. Mosca and R. de Wolf. Quantum lower bounds by polynomials, *in* "Proc. 39th Symp. on Found. of Comp. Sci.," (1998), 352-361.
4. C. Bennett, E. Bernstein, G. Brassard and U. Vazirani. Strengths and weaknesses of quantum computing, *SIAM J. Comp.* **26**(5) (1997), 1510-1523.

5. E. Bernstein & U. Vazirani. Quantum complexity theory, *SICOMP* **26**(5) (1997), 1411-1473.
6. M. Boyer, G. Brassard, P. Høyer, A. Tapp. Tight bounds on quantum searching, *Forschritte der Physik* **46**(4-5) (1998), 493-505.
7. G. Brassard, P. Høyer and A. Tapp. Quantum counting, *in* "Proc. 25th Int. Conf. on Automata, Languages and Programming" (1998), 820-831.
8. G. Brassard and P. Høyer. An exact quantum polynomial-time algorithm for Simon's problem, *in* "Proc. Fifth Israeli Symp. on Theory of Comp. and Systems" (1997), 12-23.
9. N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan and C. Tamon. Oracles and queries that are sufficient for exact learning, *J. Comput. Syst. Sci.* **52**(3) (1996), 421-433.
10. N. Bshouty and J. Jackson. Learning DNF over the uniform distribution using a quantum example oracle, *SIAM J. Comp.* **28**(3) (1999), 1136-1153.
11. H. Buhrman, R. Cleve and A. Wigderson. Quantum vs. classical communication and computation, *in* "Proc. 30th Symp. on Theory of Comp." (1998), 63-68.
12. R. Cleve. An introduction to quantum complexity theory, *to appear in* "Collected Papers on Quantum Computation and Quantum Information Theory," ed. by C. Macchiavello, G.M. Palma and A. Zeilinger.
13. D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation, *Proc. Royal Society of London A,* **439** (1992), 553-558.
14. R. Gavaldà. The complexity of learning with queries, *in* "Proc. Ninth Structure in Complexity Theory Conference" (1994), 324-337.
15. O. Goldreich, S. Goldwasser and S. Micali. How to construct random functions, *J. ACM* **33**(4) (1986), 792-807.
16. L. K. Grover. A fast quantum mechanical algorithm for database search, *in* "Proc. 28th Symp. on Theory of Comp." (1996), 212-219.
17. J. Håstad, R. Impagliazzo, L. Levin and M. Luby. A pseudorandom generator from any one-way function, *SIAM J. Comp.* **28**(4) (1999), 1364-1396.
18. T. Hegedűs. Generalized teaching dimensions and the query complexity of learning, *in* "Proc. Eighth Conf. on Comp. Learning Theory," (195), 108-117.
19. L. Hellerstein, K. Pillaipakkamnatt, V. Raghavan and D. Wilkins. How many queries are needed to learn? *J. ACM* **43**(5) (1996), 840-862.
20. M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata, *J. ACM* **41**(1) (1994), 67-95.
21. M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudrandom functions, *SIAM J. Comp.* **17**(2) (1988), 373-386.
22. J. Patarin. How to construct pseudorandom and super pseudorandom permutations from one single pseudorandom function, *in* "Adv. in Crypt. – EUROCRYPT '92" (1992), 256-266.
23. J. Pierpzyk. How to construct pseudorandom permutations from single pseudorandom functions. *in* "Adv. in Crypt. – EUROCRYPT '90" (1990), 140-150.
24. J. Pierpzyk and B. Sadeghiyan. A construction for super pseudorandom permutations from a single pseudorandom function. *in* "Adv. in Crypt. – EUROCRYPT '92" (1992), 267-284.
25. J. Preskill. Lecture notes on quantum computation (1998). Available at `http://www.theory.caltech.edu/people/preskill/ph229/`
26. R. Servedio and S. Gortler. Quantum versus classical learnability, *to appear in* "Proc. 16th Conf. on Comput. Complex." (2001).
27. P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM J. Comp.* **26**(5) (1997), 1484-1509.

28. D. Simon. On the power of quantum computation, *SIAM J. Comp.* **26**(5) (1997), 1474-1483.
29. A. Yao. Theory and applications of trapdoor functions, *in* "Proc. 23rd FOCS" (1982), 80-91.
30. A. Yao. Quantum circuit complexity, *in* "Proc. 34th FOCS" (1993), 352-361.
31. C. Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A* **60** (1999), 2746–2751.

A Proof of Theorem 3

We say that a polynomial-time deterministic algorithm $G : \{0,1\}^n \to \{0,1\}^{2n}$ is a *pseudorandom generator* if for all polynomials Q, all p.p.t. algorithms A, and all sufficiently large n, $|\Pr_{z\in\{0,1\}^n}[A(G(z)) \text{ outputs } 1] - \Pr_{z\in\{0,1\}^{2n}}[A(z) \text{ outputs } 1]| < \frac{1}{Q(n)}$. Thus a pseudorandom generator is an efficient algorithm which converts an n-bit random string into a $2n$-bit string which "looks random" to any polynomial-time algorithm. Håstad et al. [17] have shown that pseudorandom generators exist if any one-way function exists.

For G a pseudorandom generator and $s \in \{0,1\}^n$ we write $G_0(s)$ to denote the first n bits of $G(s)$ and $G_1(s)$ to denote the last n bits of $G(s)$. For $x, s \in \{0,1\}^n$ let $f_s : \{0,1\}^n \to \{0,1\}^n$ be defined as $f_s(x) \stackrel{\text{def}}{=} G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_2}(G_{x_1}(s)))\cdots))$. In [15] it is shown that $\{f_s\}$ is a pseudorandom function family. We now show that the family $\{g_s\}$ defined by $g_s(x) \stackrel{\text{def}}{=} f_s(x) \oplus f_s(x \oplus s)$ is pseudorandom.

Let $\mathcal{F}'_n$ be the following probability distribution over functions from $\{0,1\}^n$ to $\{0,1\}^n$: a function F' is drawn from $\mathcal{F}'_n$ by drawing a random function F from $\mathcal{F}_n$, drawing a random string $s \in \{0,1\}^n$, and letting F' be the function defined as $F'(x) = F(x) \oplus F(x \oplus s)$. Theorem 3 follows from the following two lemmas:

Lemma 9. *For all polynomials Q, all p.p.t. oracle algorithms M, and all sufficiently large n,* $\left|\Pr_{F\in\mathcal{F}_n}[M^F \text{ outputs } 1] - \Pr_{F'\in\mathcal{F}'_n}[M^{F'} \text{ outputs } 1]\right| < \frac{1}{Q(n)}$.

Proof. Consider an execution of M with an oracle $F' \in \mathcal{F}'_n$ defined by $F'(x) = F(x) \oplus F(x \oplus s)$. Let $S = \{x^1, \ldots, x^t\} \subset \{0,1\}^n$ be the set of strings which M uses as queries to F'. We say that M *finds* s if $x^i = x^j \oplus s$ for some $x^i, x^j \in S$. If M does not find s, then the distribution of answers which M receives from F' is identical to the distribution which M would receive if it were querying a random function $F \in \mathcal{F}_n$, since in both cases each distinct query is answered with a uniformly distributed n-bit string. Thus the left side of the inequality above is at most $\Pr[M \text{ finds } s]$. A simple inductive argument given in the proof of Theorem 3.3 of [28] shows that this probability is at most $\sum_{k=1}^{t}(k/(2^n - (k-2)(k-1)/2))$. Since M is polynomial-time, t is at most $\text{poly}(n)$ and the lemma follows. □

Lemma 10. *For all polynomials Q, all p.p.t. oracle algorithms A, and all sufficiently large n,* $\left|\Pr_{F'\in\mathcal{F}'_n}[M^{F'} \text{ outputs } 1] - \Pr_{s\in\{0,1\}^n}[M^{g_s} \text{ outputs } 1]\right| < \frac{1}{Q(n)}$.

Proof. We require the following fact which is due to Yao [29]:

Fact 2 *Let G be a pseudorandom generator, let $q(n)$ and $Q(n)$ be polynomials, and let M_* be a p.p.t. algorithm which takes as input $q(n)$ strings each of length $2n$ bits. Then for all sufficiently large n we have $|p_n^G - p_n^U| < \frac{1}{Q(n)}$, where p_n^U is the probability that M_* outputs 1 on input $q(n)$ random strings in $\{0,1\}^{2n}$ and p_n^G is the probability that M_* outputs 1 on input $q(n)$ strings each of which is obtained by applying G to a random string from $\{0,1\}^n$.*

We prove Lemma 10 by contradiction; so suppose that there exists a p.p.t. oracle algorithm M and a polynomial Q such that for infinitely many values of n,

$$\left| \Pr_{F' \in \mathcal{F}'_n}[M^{F'} \text{ outputs } 1] - \Pr_{s \in \{0,1\}^n}[M^{g_s} \text{ outputs } 1] \right| \geq \frac{1}{Q(n)}. \tag{3}$$

We will show that there is a p.p.t. algorithm M_* which contradicts Fact 2.

As in the proof in [15] that $\{f_s\}$ is a pseudorandom function family, we use a so-called "hybrid" argument. Consider the following algorithms A_i ($i = 0, 1, \ldots, n$), each of which defines a mapping from $\{0,1\}^n$ to $\{0,1\}^n$ and hence could conceivably be used as an oracle to answer M's queries. Conceptually, each algorithm A_i contains a full binary tree of depth n in which the root (at depth 0) is labeled with a random n-bit string s; if $i > 0$ then each node at depth i is also labeled with an independently chosen random n-bit string. Each node at depth $j > i$ also has an n-bit label determined as follows: if node v has label z then the left child of v has label $G_0(z)$ and the right child of v has label $G_1(z)$. Each node in the tree has an *address* which is a binary string: the root's address is the empty string, and if a node has address $\alpha \in \{0,1\}^*$ then its left child has address $\alpha 0$ and its right child has address $\alpha 1$ (so each leaf has a different n-bit string as its address). Let $L(x)$ denote the label of the node whose address is x. Algorithm A_i answers a query $x \in \{0,1\}^n$ with the n-bit string $L(x) \oplus L(x \oplus s)$.

(Note that algorithm A_i need not precompute any leaf labels. Instead, A_i can run in poly(n) time at each invocation by randomly choosing s once and for all the first time it is invoked and labeling the necessary portion of the tree "on the fly" at each invocation by choosing random strings for the depth-i nodes as required and computing descendents' labels as described above. A_i must store the random strings which it uses to label depth-i nodes so as to maintain consistency over successive invocations.)

For $i = 0, 1, \ldots, n$ let $p_n^i = \Pr[M^{A_i} \text{ outputs } 1]$, i.e. the probability that M outputs 1 if its oracle queries are answered by algorithm A_i. Let $p_n^g = \Pr_{s \in \{0,1\}^n}[M^{g_s} \text{ outputs } 1]$ and $p_n^{F'} = \Pr_{F' \in \mathcal{F}'_n}[M^{F'} \text{ outputs } 1]$. We have that $p_n^0 = p_n^g$ since algorithm A_0 behaves exactly like an oracle for g_s where s is a random n-bit string. We also have that $p_n^n = p_n^{F'}$ since algorithm A_n behaves exactly like an oracle for $F' \in \mathcal{F}'_n$. Inequality (3) thus implies that $|p_n^0 - p_n^n| \geq 1/Q(n)$ for infinitely many values of n.

Now we describe the algorithm M_* which distinguishes between sets of strings. Let $q(n)$ be a polynomial which bounds the running time of M on inputs of length n (so M makes at most $q(n)$ oracle queries given access to an

oracle from $\{0,1\}^n$ to $\{0,1\}^n$). The algorithm M_* takes as input a set U_n of $2q(n)$ strings of length $2n$. M_* works by first selecting a uniform random value $0 \leq i \leq n-1$ and a uniform random string $s \in \{0,1\}^n$. M_* then runs algorithm M, answering M's oracle queries as follows (there are two cases depending on whether or not the prefix $s_1 \ldots s_i$ is 0^i):

- **Case 1: $s_1 \ldots s_i \neq 0^i$.** Let $x = x_1 \ldots x_n$ be the query string. If no earlier query string had prefix $x_1 \ldots x_i$ or $(x_1 \oplus s_1) \ldots (x_i \oplus s_i)$, then M_* takes the next two $2n$-bit strings from U_n; call these strings $u^1 = u^1_0 u^1_1$ and $u^2 = u^2_0 u^2_1$ where $|u^i_j| = n$. M_* stores the four pairs $(x_1 \ldots x_i 0, u^1_0)$, $(x_1 \ldots x_i 1, u^1_1)$, $((x_1 \oplus s_1) \ldots (x_i \oplus s_i)0, u^2_0)$, $((x_1 \oplus s_1) \ldots (x_i \oplus s_i)1, u^2_1)$ and answers with the string

$$G_{x_n}(G_{x_{n-1}}(\ldots G_{x_{i+2}}(u^1_{x_{i+1}})\ldots)) \oplus$$
$$G_{x_n \oplus s_n}(G_{x_{n-1} \oplus s_{n-1}}(\ldots G_{x_{i+2} \oplus s_{i+2}}(u^2_{x_{i+1} \oplus s_{i+1}})\ldots)). \qquad (*)$$

  Otherwise, if an earlier query string had prefix $x_1 \ldots x_i$ or $(x_1 \oplus s_1) \ldots (x_i \oplus s_i)$, then instead M_* retrieves the two previously stored pairs $(x_1 \ldots x_i x_{i+1}, u^1_{x_{i+1}})$ and $((x_1 \oplus s_1) \ldots (x_i \oplus s_i)(x_{i+1} \oplus s_{i+1}), u^2_{x_{i+1} \oplus s_{i+1}})$ and answers with $(*)$ as above.
- **Case 2: $s_1 \ldots s_i = 0^i$.** Let $x = x_1 \ldots x_n$ be the query string. If no earlier query string had prefix $x_1 \ldots x_i$, then M_* takes the next $2n$-bit string from U_n; call this string $u = u_0 u_1$ where $|u_0| = |u_1| = n$. M_* stores the two pairs $(x_1 \ldots x_i 0, u_0), (x_1 \ldots x_i 1, u_1)$ and answers with

$$G_{x_n}(G_{x_{n-1}}(\ldots G_{x_{i+2}}(u_{x_{i+1}})\ldots)) \oplus$$
$$G_{x_n \oplus s_n}(G_{x_{n-1} \oplus s_{n-1}}(\ldots G_{x_{i+2} \oplus s_{i+2}}(u_{x_{i+1} \oplus s_{i+1}})\ldots)). \qquad (**)$$

  Otherwise, if an earlier query string had prefix $x_1 \ldots x_i$, then M_* retrieves the two pairs $(x_1 \ldots x_i x_{i+1}, u_{x_{i+1}})$ and $(x_1 \ldots x_i (x_{i+1} \oplus s_{i+1}), u_{x_{i+1} \oplus s_{i+1}})$ (these two pairs are the same if $s_{i+1} = 0$) and answers with $(**)$ as above.

The crucial properties of algorithm M_*, which are straightforwardly verified, are the following: If each string in U_n is generated by applying G to a random n-bit string (scenario 1), then M_* simulates a computation of M with oracle A_i. On the other hand, if each string in U_n is chosen uniformly from $\{0,1\}^{2n}$ (scenario 2), then M_* simulates a computation of M with oracle A_{i+1}.

It is easy to see now that in scenario 1 we have $\Pr[M_* \text{ outputs } 1] = \sum_{i=0}^{n-1} p^i_n/n$ while in scenario 2 we have $\Pr[M_* \text{ outputs } 1] = \sum_{i=1}^{n} p^{i+1}_n/n$. These two probabilities differ by $(1/n) \cdot |p^0_n - p^n_n|$, which is at least $1/nQ(n)$ for infinitely many values of n. Now by Fact 2 the existence of M_* contradicts the fact that G is a pseudorandom generator, and the lemma is proved. □

Author Index